ERPLAB TOOLBOX TUTORIAL

Version *Beta1.1.9*

1 July 2010

Tutorial written by Steve Luck, Stan Huang, and Javier Lopez-Calderon
ERPLAB Toolbox core designed by Javier Lopez-Calderon and Steve Luck

Important note: Because this is a first release, bugs are inevitable. In some cases, errors will occur leading to a message that instructs you to report the error to the EEGLAB developers. If this happens, please report the error to us at erplab@erpinfo.org and not to the EEGLAB developers. ERPLAB has primarily been tested using EEG collected with a Biosemi ActiveTwo System, along with a smaller amount of testing using EEG collected with Neuroscan and EGI systems. Consequently, you should be particularly careful when using this version of ERPLAB with data collected with systems other than the Biosemi ActiveTwo.

ERPLAB Toolbox is in a period of rapid development. This version is designed to provide most basic ERP processing functions; enhanced versions of these functions and additional functions will be forthcoming over the next several years. Feel free to make your own modifications and additions to the toolbox, and send them to us if you think they would be of value to other users.

Some features are present in this version that have not yet been documented, and some features may have changed since this document was released. In addition, some of the screen shots shown in this document may reflect previous versions and may not conform with the text. The documentation, like the software, is a work in progress. Note: This tutorial was written for a beta release that preceded the official public version 1.0.0 release, and a few small changes were made for version 1.0.0. We will provide an updated tutorial soon.

The goal of this tutorial is to provide step-by-step instructions for the processing of a typical data set. For an overview of ERPLAB Toolbox and detailed information about each specific function, see the ERPLAB User's Manual. A *Frequently Asked Questions* document is also available.

We have set up three web-based discussion groups. One is designed for people to ask questions and give and receive advice about how to use ERPLAB Toolbox. Another is designed for reporting bugs and describing work-arounds. The third is designed to give users the opportunity to make suggestions about modifications and new features. They can be accessed at http://erpinfo.org/erplab/erplab-discussion/erplab-discussion-forum/ (you will need to join the erpinfo.org web site to access these discussion groups). There is also an email list, erplab@erpinfo.org, that you can use to communicate with the developers and with other users. If you would like to join this list, you should go to http://mail01.cmb.ucdavis.edu/mailman/listinfo/erplab.

Please keep in mind that this is **FREE** software, and we do not have the resources to provide the level of support that commercial software vendors can provide.

Table of Contents

1. OVERVIEW OF TUTORIAL 5

2. GETTING STARTED 5

3. A BRIEF DESCRIPTION OF THE EXAMPLE EXPERIMENT 7

4. ADDING CHANNEL LOCATIONS TO YOUR DATASET 8

5. BACKGROUND CONCEPTS: DATASETS, ERPSETS, AND BINS 11

6. CREATING AN EVENTLIST 12

7. CREATING BIN-BASED EEG EPOCHS 18

8. ARTIFACT DETECTION 19

9. CREATING AVERAGED ERPS 25

10. PLOTTING AVERAGED ERP WAVEFORMS 27

11. FILTERING EEG AND ERPS 33

12. COMBINING ERP WAVEFORMS WITH BIN OPERATIONS 37

12.1. MODIFYING AN EXISTING ERPSET (RECURSIVE UPDATING) 37

12.2. CREATING A NEW ERPSET (INDEPENDENT TRANSFORMATIONS) 39

13. CREATING AND MODIFYING CHANNELS WITH CHANNEL OPERATIONS 40

13.1. CREATING A BIPOLAR VEOG CHANNEL BY UPDATING THE CURRENT DATASET 41

13.2. INSERTING A BIPOLAR VEOG CHANNEL BY CREATING A NEW DATASET 42

13.3. RE-REFERENCING TO THE AVERAGE OF THE LEFT AND RIGHT EARLOBES 44

13.4. RE-REFERENCING TO THE AVERAGE OF ALL SCALP SITES 45

14. ASSIGNING EVENTS TO BINS WITH BINLISTER 46

15. EXPORTING AND IMPORTING EVENTLISTS TO COMBINE ARTIFACT REJECTION AND ARTIFACT CORRECTION 50

1. Overview of Tutorial

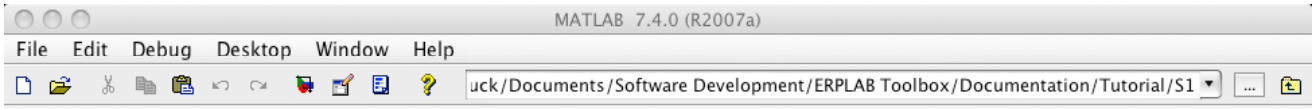
This tutorial takes you step by step through the analysis of a set of ERP data from an actual published experiment (Luck et al., 2009, *Psychophysiology*, *46*: 776-786). It focuses on giving you experience with the specific steps involved in analyzing a simple prototypical ERP experiment. For details of the functions used here, see the ERPLAB Toolbox User's Manual.

2. Getting Started

We assume that you have already installed EEGLAB and ERPLAB, as described in the ERPLAB Toolbox User's Manual.

To begin this tutorial, we recommend that you start afresh with EEGLAB and ERPLAB. To do this, close the main EEGLAB window and type "clear all" and then "eeglab" in the Matlab command window (each Matlab command should be followed with the **return** or **enter** key). It is a good idea to follow this procedure when you encounter bugs or unexplained problems with EEGLAB and/or ERPLAB.

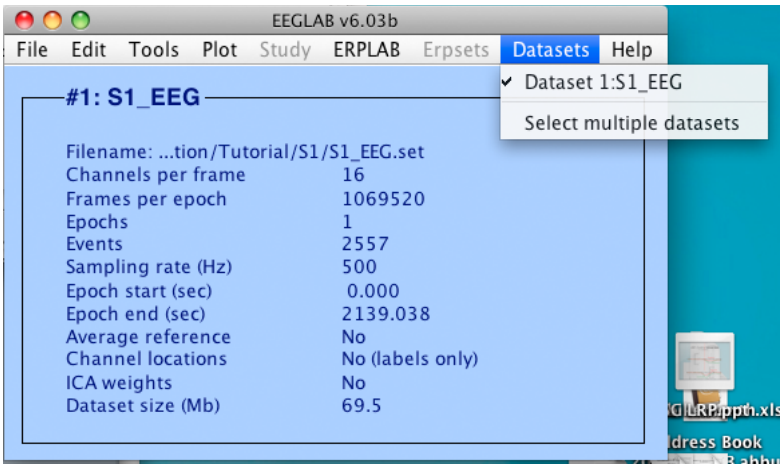
Next you need to put the various tutorial files on your hard drive, preferably in a folder named **Tutorial**. The data from the first subject we will be processing is in a subfolder named **S1**, and you should make this folder the current directory in the Matlab command window. At the top of the Matlab command window, there is a drop-down menu displaying the current directory (see screenshot below). Click on the **browse** button (labeled with '…') to the right of the current directly and choose the **S1** folder. The top of the window will then look something like this (although the details depend on what operating system you are using):


MATLAB 7.4.0 (R2007a)
File Edit Debug Desktop Window Help
uck/Documents/Software Development/ERPLAB Toolbox/Documentation/Tutorial/S1

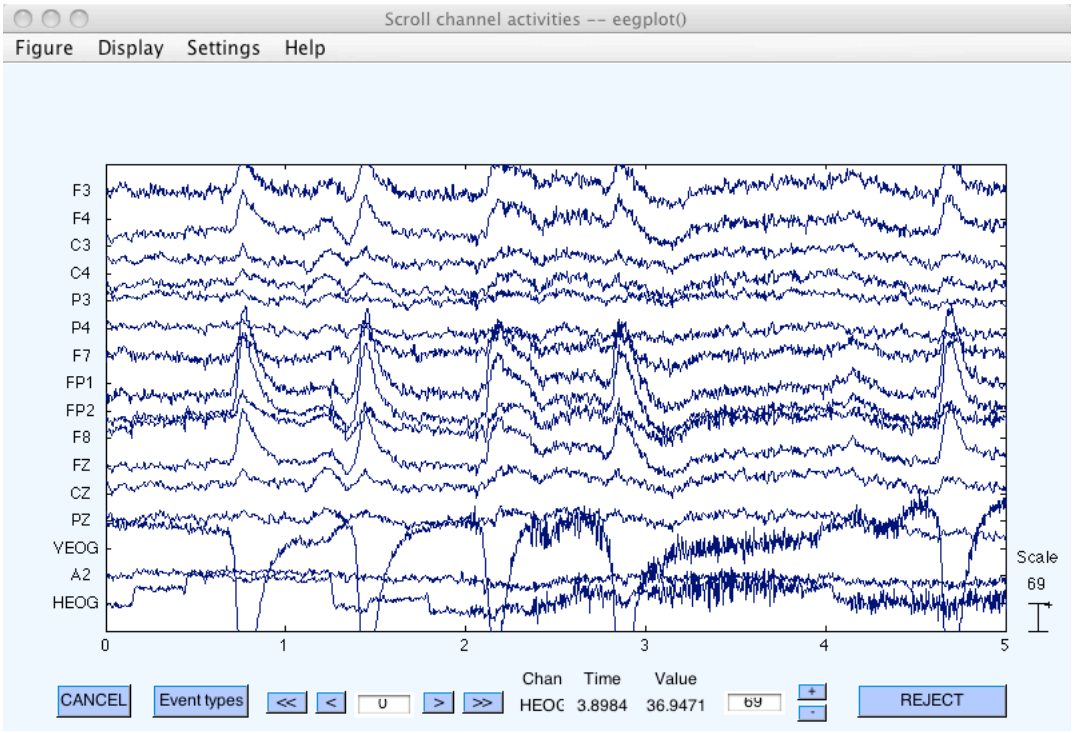
Next you will need to load the raw EEG data from the first subject into EEGLAB. To do this, select **File > Load existing dataset** from the EEGLAB GUI, and open the file **S1_EEG.set**. This file is what EEGLAB calls a "dataset", and it stores EEG and related information. The file is fairly large, and EEGLAB will probably take several seconds to load it. EEGLAB prints some information in the command window while it is processing the file, finishing with "Done" when the file has finished loading. (Both EEGLAB and ERPLAB print useful information such as this in the command window, and you should get in the habit of keeping an eye on this window.)

Hint: Using EEGLAB and ERPLAB requires a little bit of patience. If you try to start the next procedure before the current procedure is done, you will cause problems that will ultimately slow you down. This is especially true of procedures that create or load datasets. So just take a sip of coffee and wait until you see "Done" in the command window before you take the next step.

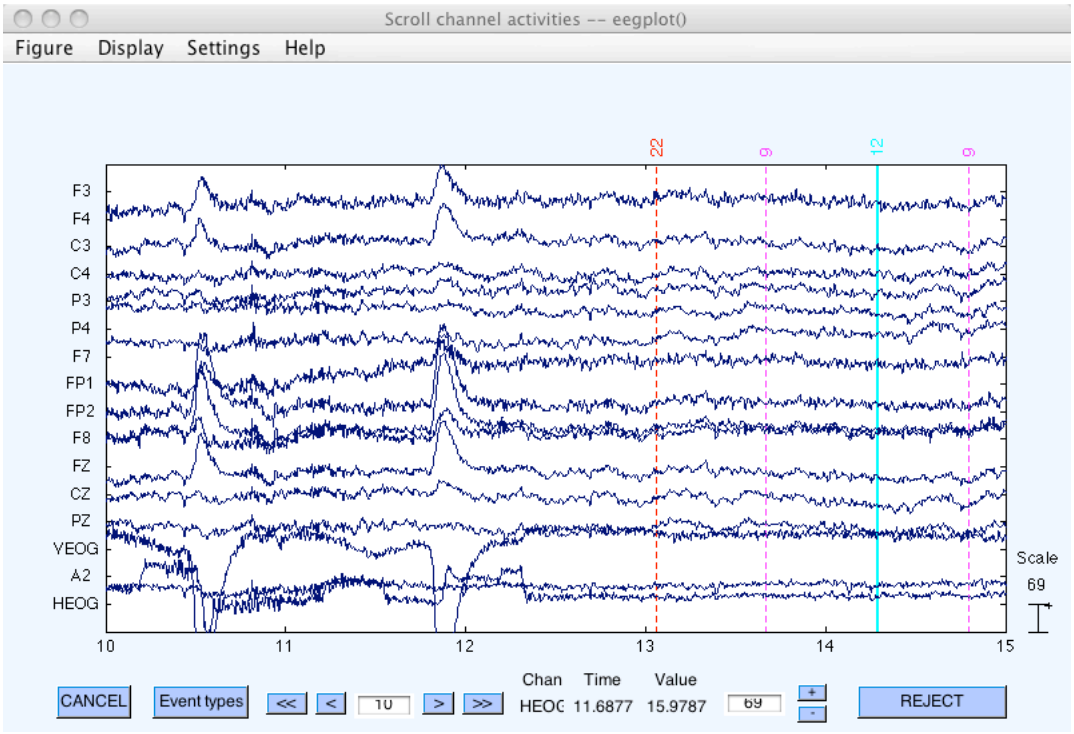
When the dataset has finished loading, it will be visible in the **Datasets** menu, as shown in the following screenshot.



You can now look at the data with **Plot > Channel data (scroll)**, as shown in the following screenshot (see the EEGLAB documentation for information about how to use this function).



If you scroll 10 seconds (by clicking the ">>" button twice), you will see vertical lines and numbers, which show the event codes (aka stimulus markers, trigger codes), as shown here:



3. A Brief Description of the Example Experiment

The initial examples come from a published experiment. In this section, we will provide a basic overview of the experiment, along with a few of the details you will need to know to analyze it. For a complete description of the experimental paradigm, see the publication ((Luck et al., 2009, *Psychophysiology*, *46*: 776-786).

This study examines the P3 wave and the lateralized readiness component (LRP) in a group of schizophrenia patients and a group of matched control subjects. This tutorial focuses on the data from 6 of the control subjects (in the folders named S1 – S6; but note that some releases contain only the data from subject S1). All subject identity information has been stripped from the data files.

In this experiment, a letter or digit was presented every 1300-1700 ms. Subjects were instructed to press a button with one hand for digits and with the other hand for letters. For a given trial block, either the letters or the digits were rare (20%) and the other category was frequent (80%). Thus, the stimulus category and the probability were counterbalanced. The probability manipulation was designed to isolate the probability-sensitive P3b component. Different event codes were used for the digits when they were rare, the digits when they were frequent, the letters when they were rare, and the letters when they were frequent. (The experiment also included a condition in which the two categories were equiprobable, but that condition is not included in this tutorial.)

The experiment was also designed to isolate the LRP by counterbalancing the stimulus category and probability with the left and right hand responses. That is, subjects responded to letters with the left hand and digits with the right hand for half of the experiment, and they responded to digits with the left hand and letters with the right hand for the other half. This is also indicated by the

event codes for each stimulus (i.e., the event code for the stimulus also indicates the appropriate response for that stimulus).

ERPLAB Toolbox makes no distinction between stimulus event codes and response event codes. This gives the user more power, but it also gives the user responsibility for using the event codes in a sensible way. The event codes for responses in this example experiment indicated whether the response was correct (event code = 9) or incorrect (event code = 8) rather than directly indicating whether a left-hand or right-hand response was made. However, the combination of the stimulus event code and the response event code makes it possible to determine whether the subject pressed the left button or the right button.

Here is a table showing the stimulus event codes.

Event Code	Category	Probability	Correct Response
11	Letter	Frequent	Left Hand
21	Digit	Rare	Right Hand
112	Letter	Rare	Left Hand
122	Digit	Frequent	Right Hand
12	Letter	Rare	Right Hand
22	Digit	Frequent	Left Hand
111	Letter	Frequent	Right Hand
121	Digit	Rare	Left Hand

The data in this experiment were recorded from a 16-bit Neuroscan Synamps system with a bandpass of 0.05-100 Hz and a sampling rate of 500 Hz. Each scalp site was recorded relative to a right-earlobe reference. Recordings were also obtained from a vertical electrooculogram (VEOG) electrode located beneath the left eye and from an electrode on the left earlobe (both referenced to the right earlobe). A horizontal electrooculogram (HEOG) signal was recorded as the potential between electrodes located just lateral to each eye.

Letters and digits were presented in an unpredictable order within each trial block, but separate blocks were used for each combination of probability response assignment. That is, there were separate trial blocks for: (1) 80% letters (left-hand response) and 20% digits (right-hand response); (2) 20% letters (left-hand response) and 80% digits (right-hand response); (3) 80% letters (right-hand response) and 20% digits (left-hand response); (4) 20% letters (right-hand response) and 80% digits (left-hand response). Each block was originally recorded in a separate file.

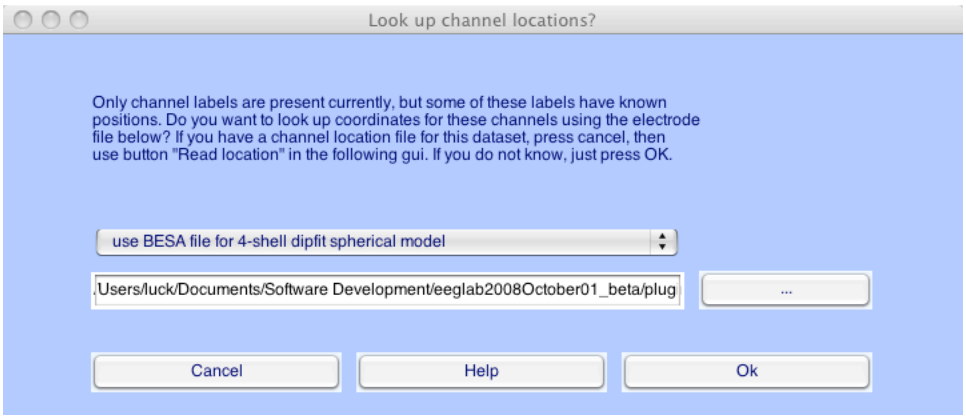
These files were imported into EEGLAB (using **File > Import data > From Neuroscant .CNT File**) and then merged together (with **File > Append datasets**). The procedure you follow will depend on your data acquisition system. Note that EEGLAB places a **boundary** event code at the boundary between the original files to mark the fact that a temporal discontinuity was present at those times. This is important, because some functions (e.g., filters) either implicitly or explicitly assume that consecutive data points in the **EEG** reflect temporally consecutive moments in time (see the section on boundary events in the ERPLAB User's Manual). The end result was saved in a file named S1_EEG.set (or S2, S3, etc., to indicate each individual subject).

4. Adding Channel Locations to your Dataset

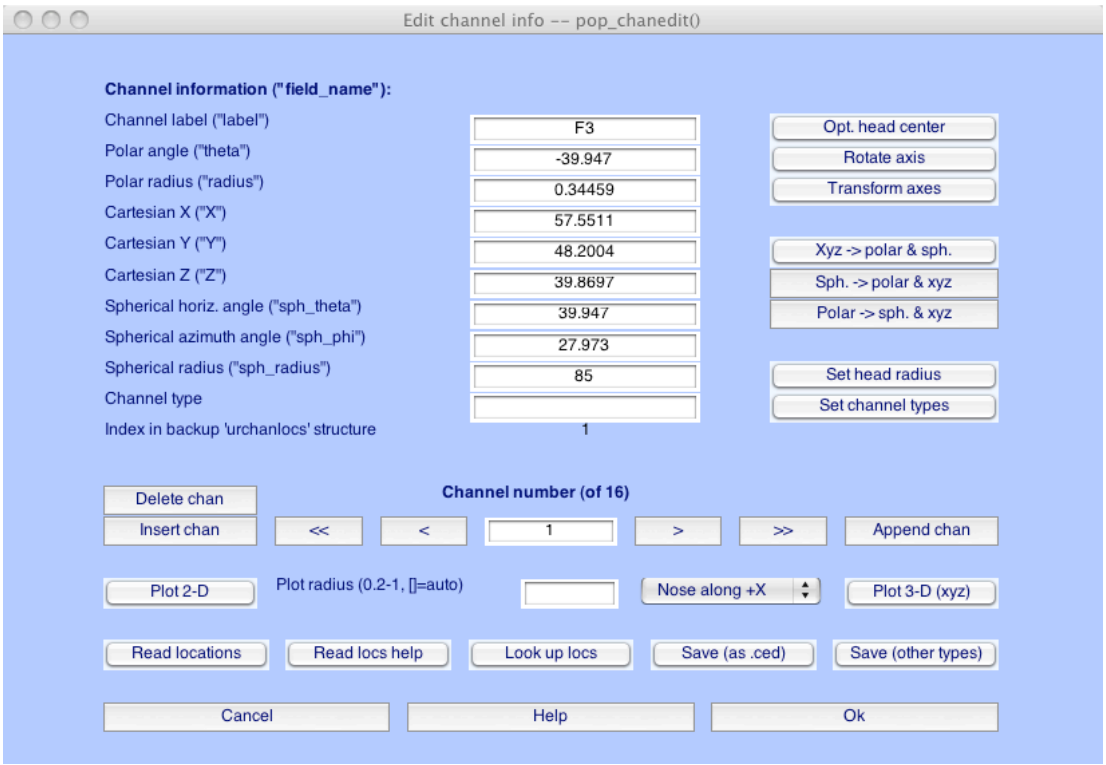
Before we start using ERPLAB's functions, we'll use one of EEGLAB's functions that will prove useful at a later stage. Specifically, we will add location coordinates for each electrode site (see the EEGLAB documentation for more information). Our original data file contained electrode labels

conforming to the International 10/20 System, and EEGLAB can automatically determine standard coordinates on the basis of these labels. These coordinates are not guaranteed to be exact, but they are a reasonable approximation for simple procedures such as plotting scalp maps. If you want exact electrode locations, you will need to use a 3-D digitizer and measure the locations for each subject.

To add coordinates based on the electrode labels, select **Edit > Channel locations**, which will bring up the window shown here:


Look up channel locations?
Only channel labels are present currently, but some of these labels have known positions. Do you want to look up coordinates for these channels using the electrode file below? If you have a channel location file for this dataset, press cancel, then use button "Read location" in the following gui. If you do not know, just press OK.
use BESA file for 4-shell dipfit spherical model
Users/luck/Documents/Software Development/eeglab2008October01_beta/plug
...
Cancel
Help
Ok

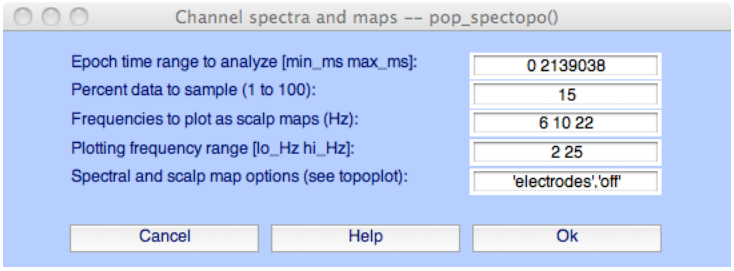
Click **Ok**, and you will see the window shown here:



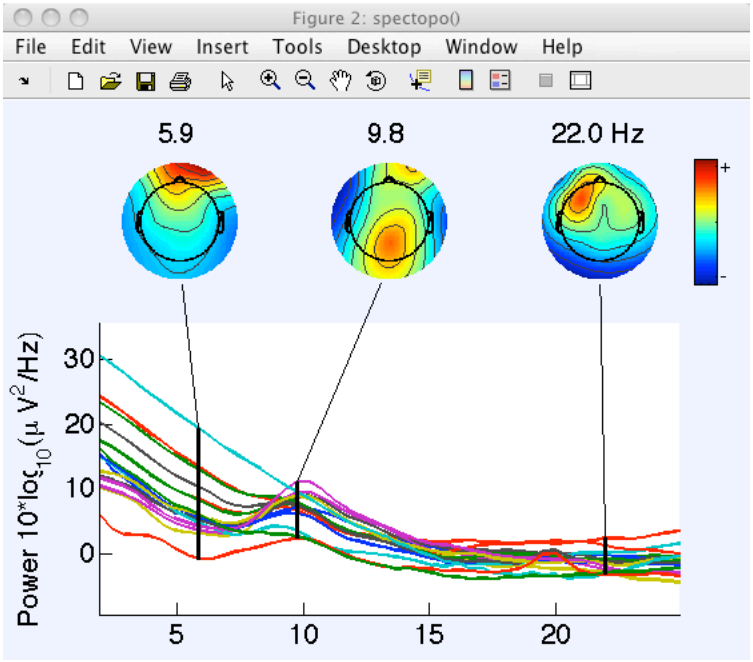
For simple data files such as the one used in this tutorial, you can accept all of the default values and simply click **Ok**. This will add the coordinates to the current dataset.

EEGLAB and ERPLAB require electrode coordinates for plotting topographic maps; you will get an error message if you try to plot a topographic map before you've added the coordinates. You can

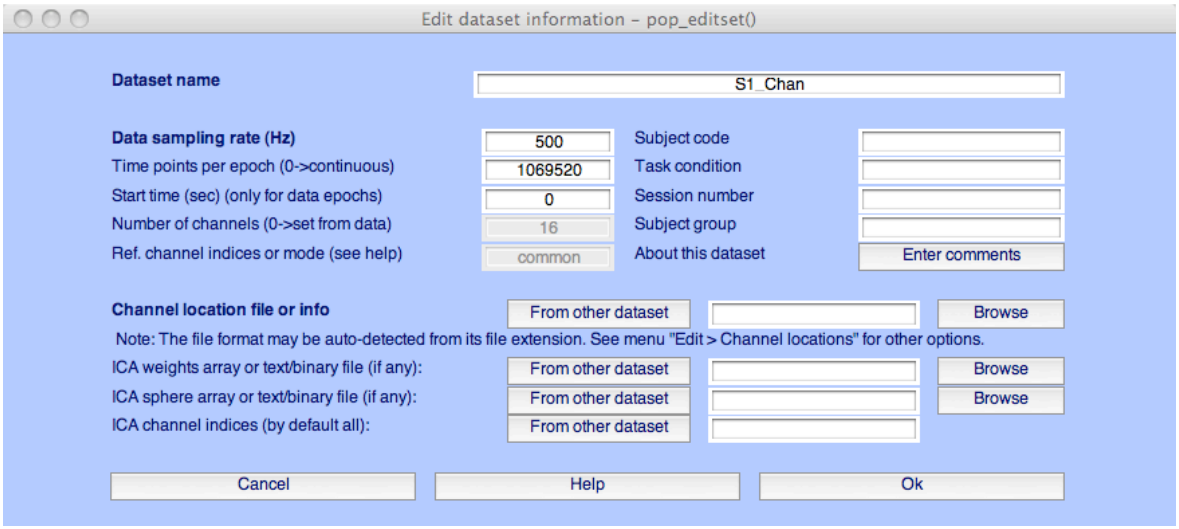
verify that the coordinates have been added by running a routine that plots topographic maps. For example, if you select **Plot > Channel spectra and maps** and select the default parameters in the window that appears, as shown in the screenshot below:



Click **OK** and you will see the following plot:



Later parts of this tutorial will make use of this dataset, so you should save it to disk. First, however, you should change the name of the dataset to **S1_Chan** so that you can differentiate between this version and the original version. To do this, select **Edit > Dataset info**. This will bring up the window shown below, and you should enter **S1_Chan** in the **Dataset name** field. Click **Ok**, and then look in the **Datasets** menu. You should now see that it is named **S1_Chan**.



You can now save the modified dataset by selecting **File > Save current dataset as**. We recommend that you give the file the same name as the dataset, but with a **.set** extension (**S1_Chan.set**).

5. Background Concepts: Datasets, ERPsets, and bins

There are a few key concepts you need to understand before we go any further. EEGLAB stores a set of EEG data in something called a *dataset*. A dataset typically stores the data from a single subject, either a single block of trials or an entire session. A dataset is maintained in memory inside EEGLAB, and it can also be saved on disk. Whenever you run a routine that changes the data in a dataset, a new dataset is created. You can see the currently available datasets in the **Datasets** menu. One dataset is currently active, and any routines that you run will typically be applied to the current dataset. When a new dataset is created, it becomes the current dataset, but you can make a different dataset active by selecting it in the **Datasets** menu. This provides a very convenient workflow.

ERPLAB defines an analogous structure called an *ERPset*, which stores a set of ERP waveforms. They can be active inside ERPLAB and/or saved to disk. The **ERPsets** menu can be used to see which ERPsets are currently loaded into ERPLAB and to change which ERPset is active.

A *bin* is a set of averaged ERP waveforms, one for each electrode site, which were created by averaging together a specific set of EEG epochs. A simple oddball experiment, for example, might have one bin for the infrequent targets and another bin for the frequent standards. However, a sophisticated experiment might have dozens of different bins, with a given bin being something like "Digits preceded by letters followed by a correct response between 200 and 1000 ms, in a condition in which digits are rare and letters are frequent." In many ERP analysis systems, a bin would be equivalent to a single averaged ERP file. However, this can lead to a huge number of different files for each subject, making it difficult to keep track of everything. In ERPLAB, an ERPset can contain an unlimited number of bins), and each data processing operation is typically applied to all of the bins in the currently active ERPset. This saves time and reduces errors.

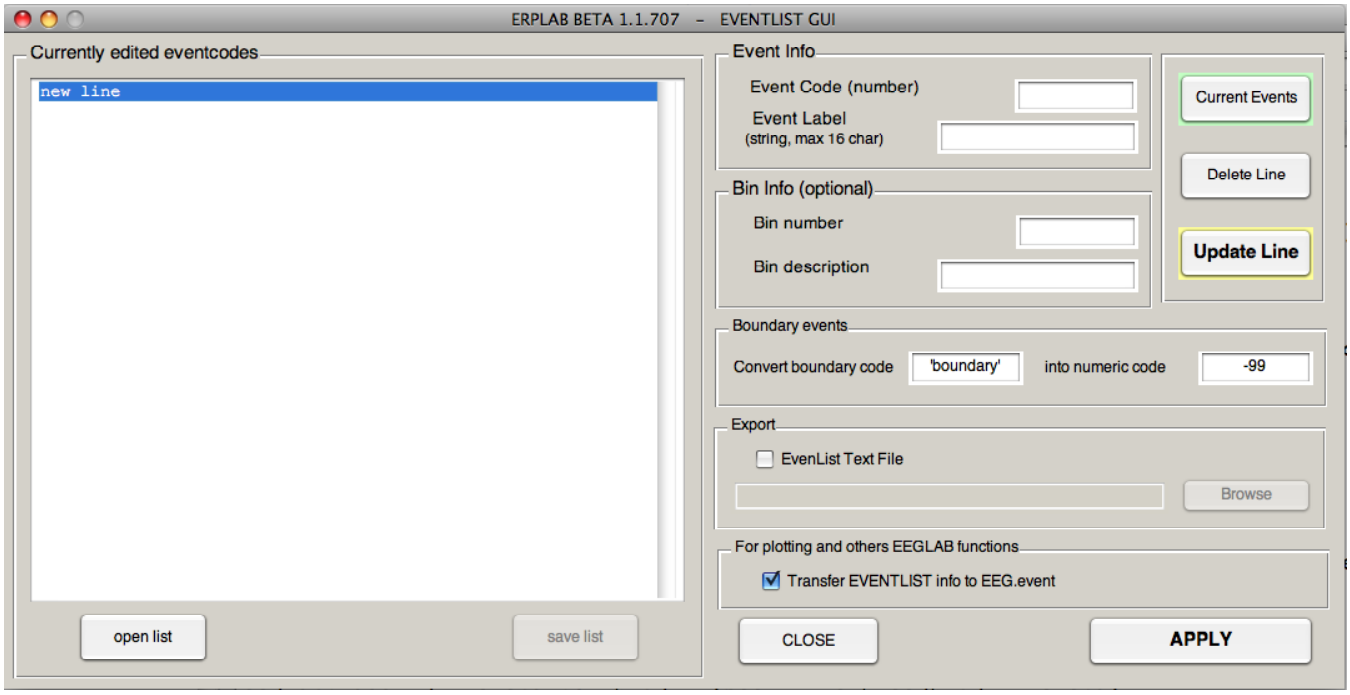
6. Creating an EventList

We are now ready to get into the ERPLAB functions. We will begin with some housekeeping, then do artifact rejection, and finally averaging.

The event codes are embedded in a dataset within EEGLAB's **EEG** structure, which is pretty complicated. To make it easier to see and manipulate the event codes, ERPLAB creates an EventList, which is a list of all of the event codes and associated information. Internally, ERPLAB stores this as an **EVENTLIST** structure (which is appended onto the **EEG** structure as **EEG.EVENTLIST**). But the EventList can also be saved in a text file, where you can easily see and manipulate the event information. If you make any changes, you can import it back into the **EEG** structure. We will show an example of this later in the tutorial.

To use ERPLAB, you must create an EventList for the **EEG** stored in your dataset. The simplest way to do this is to choose **ERPLAB > EventList > Create EEG EventList – Basic**. This is pretty self-explanatory, so we will show you how to use **ERPLAB > EventList > Create EEG EventList – Advanced**. The advanced version allows you to define a text label for each numeric event code (or vice versa), which helps to avoid the errors that often occur in experiments with many different event codes. It also allows you to define the bins that you will use to sort your EEG epochs for averaging. In most cases, you will define your bins in a later step (using something called BINLISTER), but it can be done at this step if your bins are very simple. You will use the advanced method for creating events if you want to create event labels from numeric event codes or if you want to assign the events to bins, and you will use the basic method if you don't want event labels and you will be using BINLISTER to assign events to bins.

At this point, EEGLAB and ERPLAB should be running, and S1_EEG should be the currently active dataset (you can check this in the **Datasets** menu). You should have added the electrode coordinates to the dataset in a prior step (so make sure that S1_Chan is the currently selected dataset). To create the EventList for this dataset, select **ERPLAB > EventList > Create EEG EventList – Advanced**, which will bring up the window shown here:



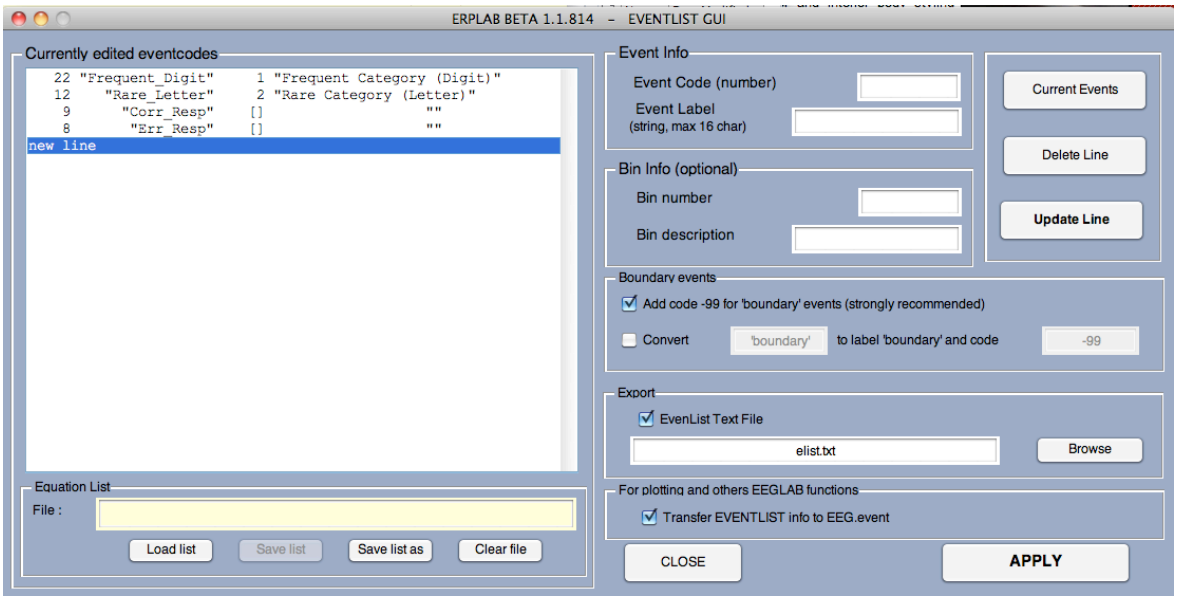
This window allows you to create a table (shown in the left portion of the window) that defines a set of numeric event codes, text-based event labels, bin numbers, and bin labels for each type of event in your experiment. The current version of ERPLAB requires that each event have a numeric event code. If your data acquisition system uses text-based labels instead of numeric codes, you must define a numeric event code for each text-based event label. If your data acquisition system uses numeric codes, you are not required to create text-based labels for each code, but it's a good idea. You can also define a bin for all (or some) of your event codes, along with a label for that bin. We will do that in this example, but it is not necessary if you will be using BINLISTER in a later step to define your bins.

To keep our example simple, we will start by defining this information for only one of the trial blocks in the example experiment. In this block, letters were rare (event code 12) and numbers were frequent (event code 22). You can use the **Event Info** area of the window to define the event codes, event labels, bin numbers, and bin descriptions for each event. For the digits, for example, you can enter **22** in the **Event Code (number)** field, **Frequent_Digit** in the **Event Label** field, **1** in the **Bin number** field, and **Frequent Category (Digit)** in the **Bin description** field (see screenshot below). If you then click **Update Line**, this definition will appear in the table at the left of the window. You can do the same thing to define the letters (event code = **12**, event label = **Rare_Letter**, bin number = **2**, bin description = **Rare Category (letters)**).

We will be computing stimulus-locked averages rather than response-locked averages in this example, so we do not need to define bins for the response event codes. However, it will still be useful to define labels for the response codes so that we will not have to remember that code 9 denotes a correct response and code 8 denotes an incorrect response. The table at the left of the window in the screenshot below shows how to define these events.

Don't forget to click **Update Line** after entering the information for each type of event or the information will not be added to the table. Also, your bin numbers must start at 1 and be consecutive with no breaks (but event codes need not be consecutive, and they can be any integer number that Matlab can represent).

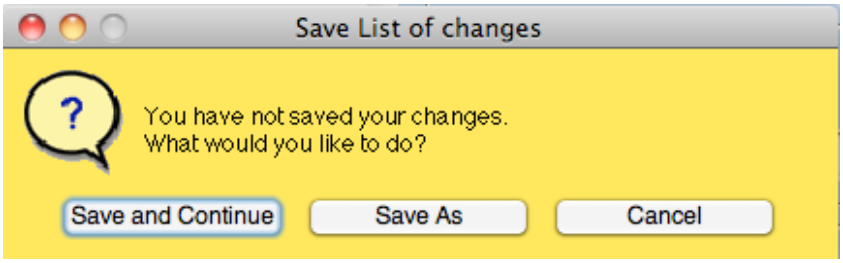
Because you will need to do this step for every subject, you can save the table of event information in a text file (by clicking **save list**) and then load it in again for the next subject (by clicking **open list**). If you want, you can load the file "event_mapping_1.txt", which contains the table of events for this example. You can edit this file with a text editor (such as the Matlab editor), which is more convenient than using the ERPLAB GUI for large tables of event information. The format is pretty obvious (you can open the event_mapping_1.txt file with a text editor to see the format).



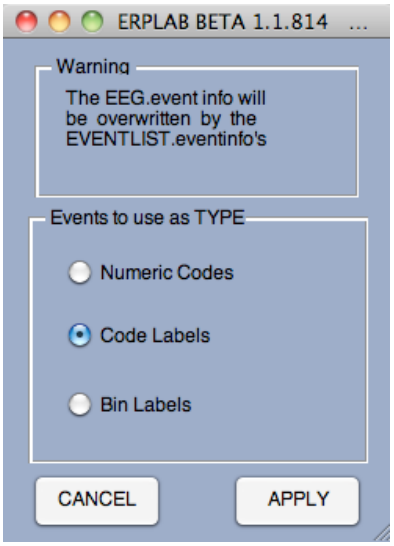
In this example, we will save the EventList in a text file to look at it later. To do this, check the box labeled **EventList Text File** and click the **Browse** button to save the file as "elist.txt". (The .txt extension is not required, but it's a good idea.)

The check box labeled **Transfer EVENTLIST info to EEG.event** should be checked. EEGLAB keeps track of event information in the **EEG.event** structure, and EEGLAB doesn't "know" anything about ERPLAB's EventList. Moreover, the **EEG.event** structure can store either a numeric event code or a text-based event label for a given event, but not both. By checking this box, you will have the opportunity to transfer your newly created labels into **EEG.event** so that they will be available for use by EEGLAB functions (such as the function for plotting the EEG data).

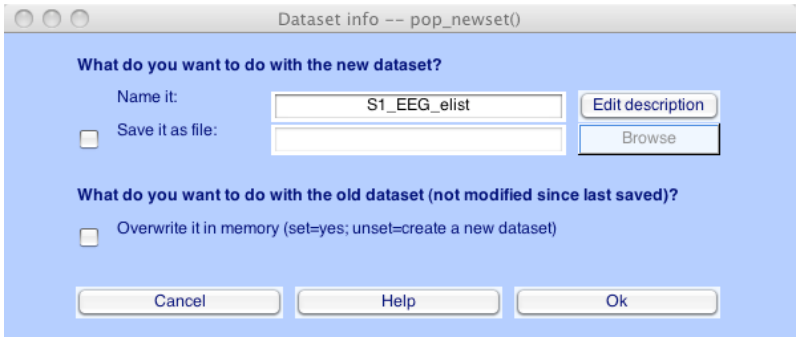
Once you have entered all of the necessary information into the window, as shown in the screenshot above, click the **APPLY** button to create the EventList and add it to the current dataset. The information about event mapping must be stored in a file. Therefore, if you have made any changes in this information, the following window will appear:



If you select **Save and Continue**, it creates the EventList and will put up the window shown in the next screenshot (assuming that you checked the **Transfer EVENTLIST info to EEG.event** box). This window allows you to choose whether to store the event codes, the event labels, or the bin numbers in the **EEG.event** structure). The default is **Code Labels**, and selecting this will allow you to see the labels for each event code when you look at the EEG using EEGLAB's **Plot > Channel data (scroll)** routine. Keep this default setting and click the **APPLY** button.

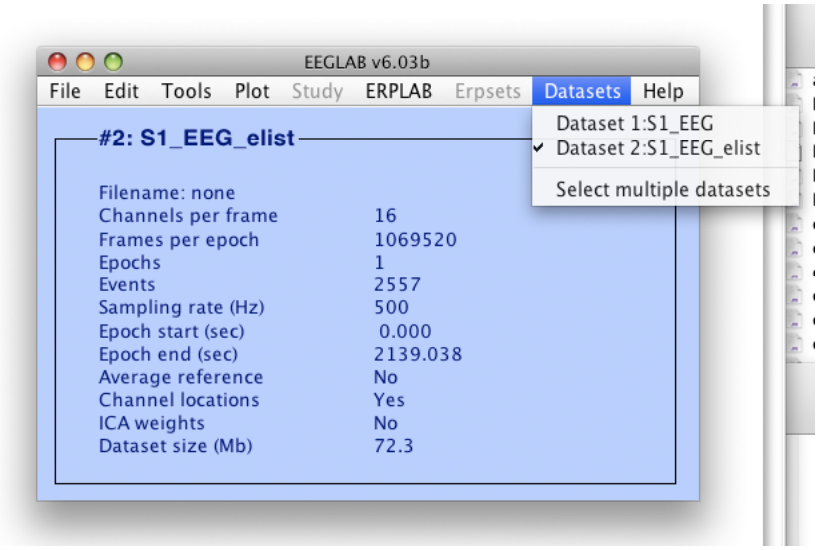


ERPLAB attaches the **EVENTLIST** structure that you just created to the current dataset, and this produces a new dataset. Any operation that creates a new dataset will put up the window shown in the next screenshot, which gives you some options for saving the new dataset. First, you need to specify a name for the new dataset. This name will be stored inside the dataset, and it will be shown in the **Datasets** menu. It is not necessarily the same as the name of the file in which the dataset is stored on disk, but we recommend that you use the same name for the file (if you save it as a file on disk). ERPLAB will automatically suggest a name for the dataset, appending a string onto the end of the name of the dataset from which the new dataset was created. When an EventList is generated, it suggests adding "_elist" onto the dataset name (going from "S1_EEG" to "S1_EEG_elist"). This is not required, but it is recommended (otherwise you will have a hard time remembering what is in a given dataset. If you check the **Save it as file** button and use the **Browse** button to select a filename, the dataset will be saved to disk. You can also choose to either keep the previous dataset in the **Datasets** menu (this is the default) or overwrite the previous dataset in this menu.

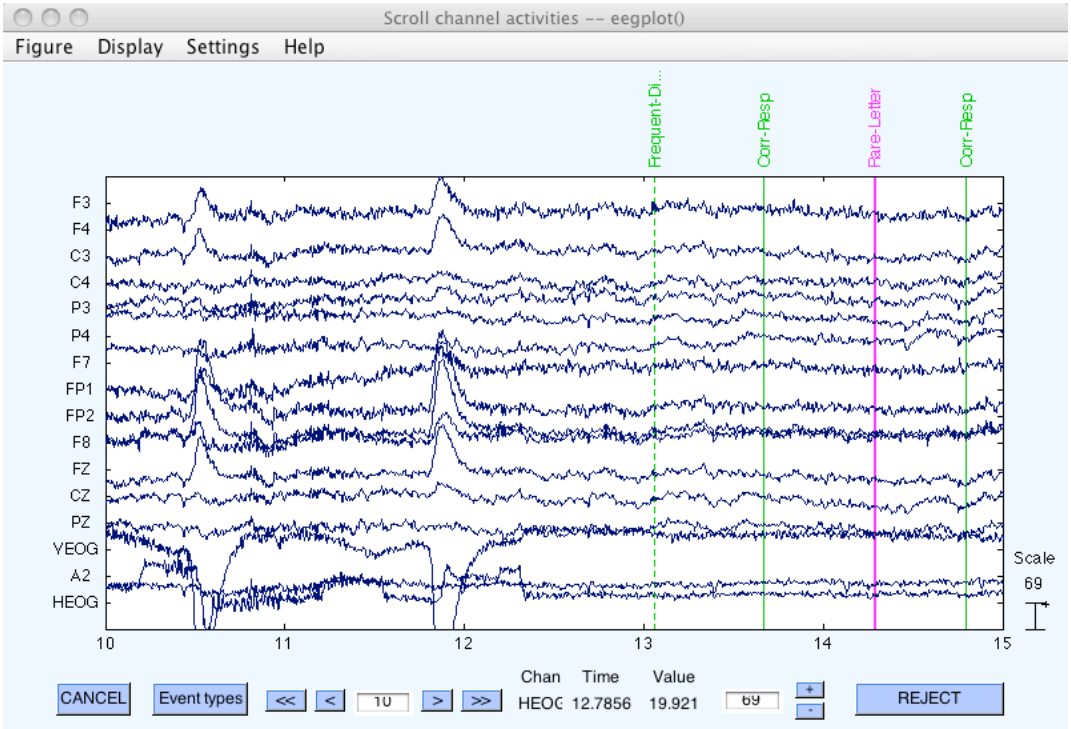

Dataset info -- pop_newset()
What do you want to do with the new dataset?
Name it:
S1_EEG_elist
Edit description
Save it as file:
Browse
What do you want to do with the old dataset (not modified since last saved)?
Overwrite it in memory (set=yes; unset=create a new dataset)
Cancel
Help
Ok

We find that it is usually a good idea to save the dataset to disk after you've gone through several steps; otherwise all your work will be lost if the program crashes. And you should obviously save to disk if you are going to quit the program and you don't want to lose what you've created. However, it usually isn't necessary to save every dataset you create – this ends up leading to a lot of big files on your hard drive. Also, we don't usually overwrite the previous dataset, because then we can't easily back up to the previous step. However, there are no hard-and-fast rules here; you will eventually develop a set of habits that make sense for your own needs.

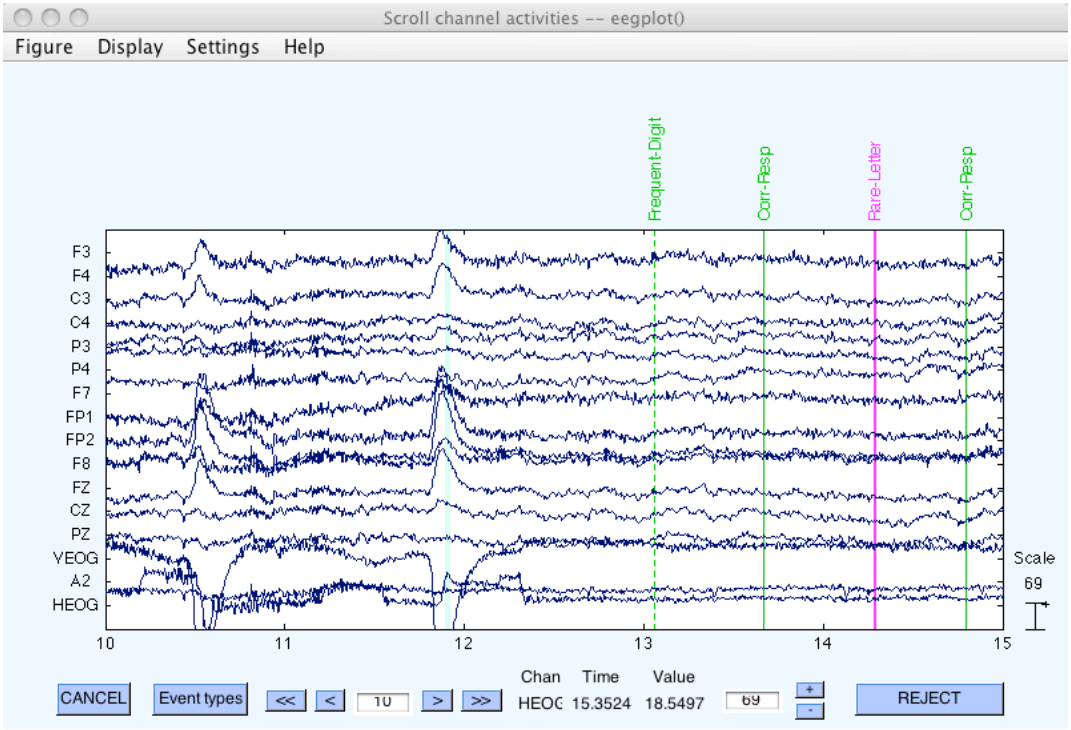
For this example, you should save the new dataset to a file, named **S1_EEG_elist.set**, but don't overwrite the old dataset. Click **Ok**, and the new dataset will be created (this may take a few seconds). Once the new dataset has been created (you will see "Done" printed in the Matlab command window), you will be able to see it in the **Datasets** menu, as shown in the next screenshot.



If you now select **Plot > Channel data (scroll)** and scroll ahead to the period starting at 10 seconds (hit the **>>** button twice), you will now see that the event code numbers that you saw before have been replaced by the event labels.



The standard EEGLAB routine only provide room for about 4 characters, but we have provided a slightly modified version that shows 12 characters for the event labels (see the installation instructions in the ERPLAB User's Manual for instructions on installing this modified plotting function). This still isn't quite enough, so some of the labels are truncated. You can modify the number of characters shown in the modified plotting function by selecting **Settings > Events > Number of characters for event labels** from within the plotting window (not from the main EEGLAB window). If you enter a width of 20 characters, the plot will look like this:



Scroll through the data for a while (and play with the plotting routine if you haven't used it before). You might want to change **Settings > Time range to display** to 30 seconds (within the plotting window). Eventually (around 120 seconds) you will see a period with no event codes and lots of artifacts. This is a short rest break about ¼ of the way through the trial block. You'll see another period like this at 255 seconds and 390 seconds. The trial block ended at approximately 525 seconds and the file ended at 535 seconds. You will see a **boundary** event label at this point, which is where the second trial block was appended after the first trial block.

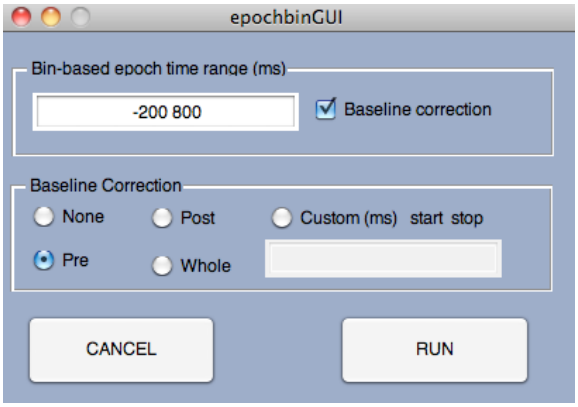
If you keep going past the **boundary** event, you will start to see some numeric event codes. In this simplified example of creating an EventList, we assigned labels only to the event codes for the stimuli that were used in the first block of trials (along with the responses for the entire session). You don't need to assign labels for all of your event codes (or even any of them). However, having the labels makes it much easier to link the signals that you see in the EEG to specific types of events (especially in experiments with dozens of different event codes).

If you're interested, you can take a look at the text version of the EventList that you created. Simply open this file with your favorite text editor (such as the Matlab editor). For details on the contents and format of this file, see the ERPLAB User's Manual. Later on we will show how you can edit this file and import it back into your EEG data.

7. Creating Bin-Based EEG Epochs

The next step after creating the EventList is usually to sort the events into different bins (in preparation for averaging). However, we already did this by assigning bins for the rare and frequent stimuli when we created the EventList.

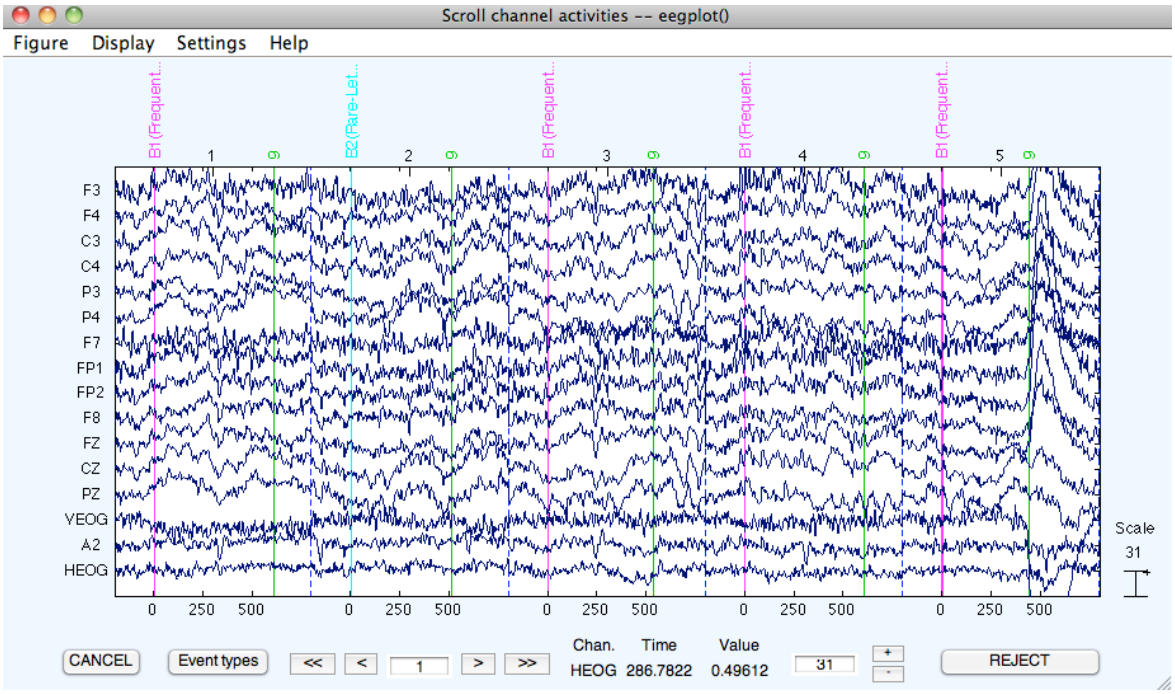
Once the events have been assigned to bins, the next step is to divide the continuous EEG into a set of fixed-length *epochs* (also known as *segments*), each of which is time-locked to an event that has been assigned to a bin. In the present example, we will extract the EEG during a period that begins 200 ms prior to the onset of a stimulus and ends at 800 ms. This will be done for each event code that has been assigned to a bin (which is, in the present example, the frequent letters and rare digits, not the responses). Note that EEGLAB has an epoching function (**Tools > Extract epochs**); you should not use this function if you are using the ERPLAB functions for processing the epochs (e.g., averaging, plotting, etc.). Instead, you should use **ERPLAB > Extract Bin-based Epochs**. If you select this function, you will see the following window:



This window allows you to select the time period for the epoch (relative to the event code). **Baseline correction** will be enabled by default, and you will almost always want to do this. The prestimulus period (**Pre**) is usually the period used for baseline correction. That is, the average voltage during the prestimulus period for a given trial is subtracted from every point in the epoch (separately for each channel). Alternatively, you can use the poststimulus interval (**Post**), the entire interval (**Whole**), or some time range of your choice (**Custom**). For this example, you should leave it as **Pre**. Click **RUN**, and it will create a new dataset with the epoched data (it will take a few seconds). The standard window for saving a new dataset will appear; you can just click **Ok** and it will name the new dataset "S1_EEG_elist_be" ("be" stands for bin-epoched). However, you should save the dataset to a file on your hard drive, because it will be needed for completing the section on combining artifact rejection and artifact correction. You can always do this later by selecting the appropriate dataset in the **Datasets** menu and then selecting **File > Save current dataset(s)** or **File > Save current dataset as**.

Make sure that this new dataset is the active dataset (by looking in the **Datasets** menu) and then select **Plot > Channel data (scroll)**. The following screen shows what it should look like. It looks like you still have continuous EEG data, but if you look more closely you will see a vertical dashed line 800 ms after (and 200 ms before) each event. This dashed line marks the end of one epoch and the beginning of the next. If you find this disconcerting, you can simply change **Settings > Time range to display** to a period of 1 second (in the plotting window).

Hint: EEGLAB routines typically ask you to enter time information in seconds, whereas ERPLAB routines typically ask for milliseconds.



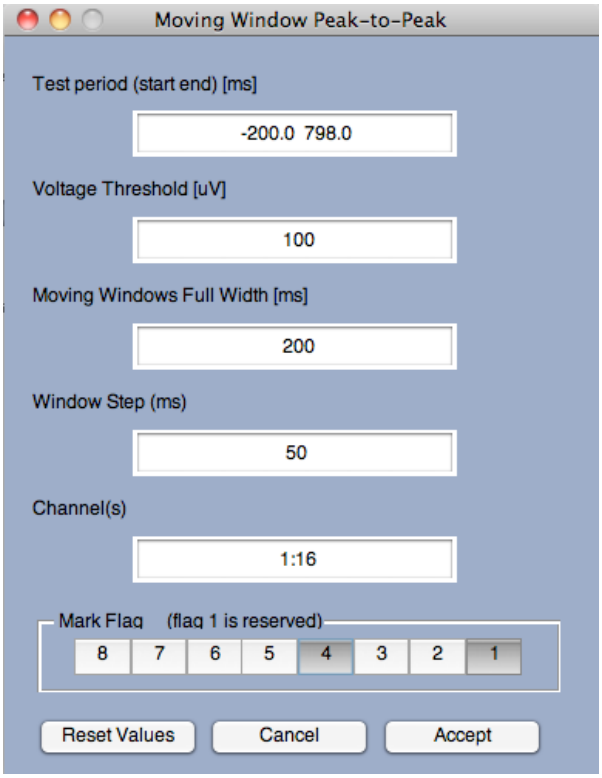
Because the epoch length in this experiment is shorter than the interval between consecutive stimuli, the epoched data will now be missing a period of time between the end of one epoch and the beginning of the next epoch. If, however, we used a 2000-ms epoch length, the last part of one epoch would contain the same EEG data as the beginning part of the next epoch.

8. Artifact Detection

The next step after epoching is usually artifact detection. ERPLAB makes a distinction between *artifact detection* and *artifact rejection*. Artifact detection is the process of marking epochs that contain artifacts. Artifact rejection is the process of excluding trials containing artifacts. The ERPLAB artifact detection routines do not throw out any data; epochs with artifacts are merely marked. Artifact rejection occurs during averaging; marked epochs are simply excluded from the averaging process.

EEGLAB contains routines for artifact detection (**Tools > Reject data epochs**) and also allows you to manually delete sections of data with artifacts from within the **Plot > Channel data (scroll)** window. The manual deletion is permanent (although you can always go back to a previous version of the data in which the data have not been deleted). Manual deletion also causes the insertion of a **boundary** event. Manual deletion is performed on continuous data, whereas artifact detection is performed on epoched data. You can, in principle, combine EEGLAB's artifact detection routines with ERPLAB's artifact detection routines, but this has not yet been thoroughly tested.

ERPLAB's artifact detection tools are mostly still in development. However, two of these tools are ready for use (peak-to-peak and step function). Make sure that your epoched dataset is active ("S1_EEG_elist_be") and select **ERPLAB > Artifact Detection > Moving Window Peak-to-Peak Threshold**. You will see the window shown in the screenshot below.



This routine finds the maximum peak-to-peak voltage within a moving window that slides across the epoch (see the ERPLAB User's Manual for details). You select the **Test period,** which is by default the entire epoch (note that the time values you selected in the epoching process have been rounded to actual sample times). However, sometimes you will want to search for artifacts only during a portion of the epoch. You cannot, of course, look for artifacts outside of the epoch period, because only the data from within the epoch period are present after the data have been epoched.

You must also choose a **Voltage threshold**. An epoch will be rejected if the peak-to-peak voltage (the difference between the largest and smallest values) exceeds this value. The appropriate value will depend on the nature of the artifact you are trying to reject and how large it is in a given subject (see Chapter 4 in *An Introduction to the Event-Related Potential Technique* for an overview of artifact detection).

You will then choose the length of the **Moving window** and the **Window step**. This routine finds the peak-to-peak voltage within a window of the length specified in the **Moving window** field. It then shifts the window by the amount specified in the **Window step** field and finds the peak-to-peak voltage within this new window. This is repeated until the end of the **Test period**, and the largest of these peak-to-peak amplitudes is compared with the **Voltage threshold**. This is an excellent way to find artifacts in which the voltage changes rapidly within a relatively short period of time (e.g., blinks, which usually peak and then subside within 200-300 ms) and not be fooled by relatively slow voltage changes.

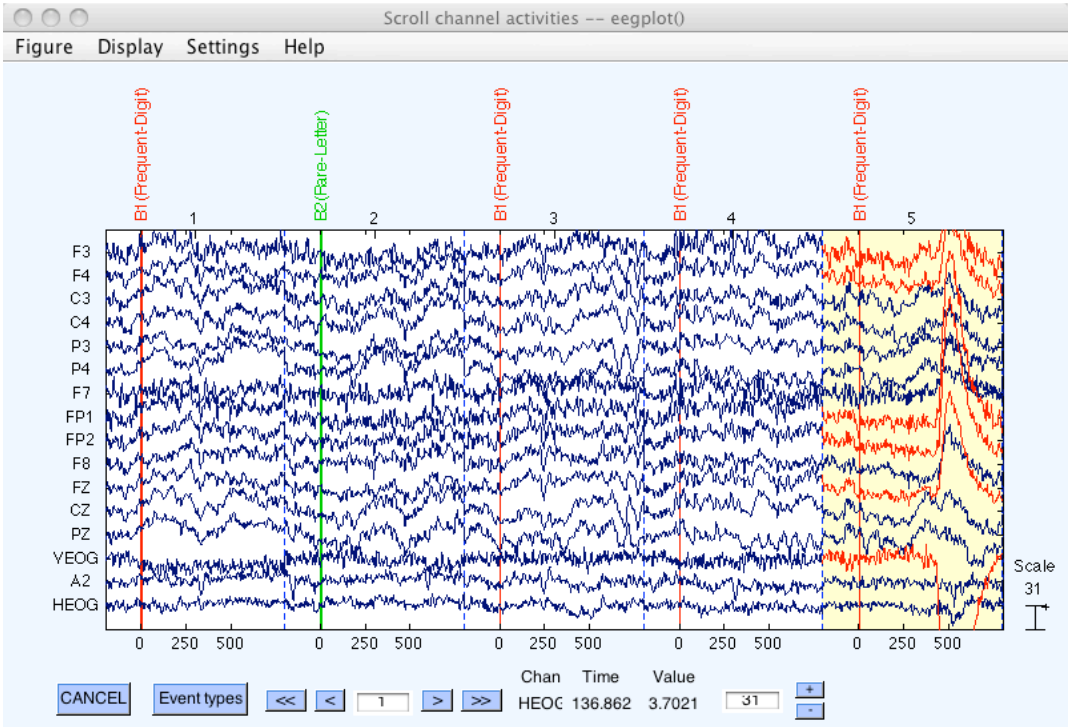
By default, artifacts will be detected in all channels. However, you can specify a subset of channels to be tested in the **Channel(s)** field.

Hint: Most windows that ask you to specify a range of values allow you to use the standard Matlab rules for numeric lists. You can simply type the values separated by spaces (e.g., "1 2 3"), or you can use a colon to specify a range (e.g., "1:3") or some combination (e.g., "1:3 5:7" to indicate 1-3 and 5-7).

To specify a regular set of values, you can specify the first value, the step size, and the last value (e.g., "1:3:7" to start with 1 and go by steps of 3 to a final value of 7 [1 4 7]).

Finally, you can specify a set of flags in the **Flag** field that will be set in the EventList whenever an epoch has been marked for rejection. This allows you to do things like perform behavioral analyses on the information in the EventList while excluding events that contain artifacts. Flag 1 is always set for events with artifacts, and you can keep track of which events had which types of artifacts by setting additional flags for each artifact rejection test you perform. For details, see the ERPLAB User's Manual.

Once you have set all of the fields as shown in the screenshot above, click **Accept**. After scanning the active dataset for artifacts (which may take some time), the total percentage of trials with artifacts detected will be shown in the Matlab command window (this is the total percentage marked for rejection, including epochs that had been marked previously). You should see that 29.7% of epochs have been marked for rejection. At this point, the ERPLAB artifact rejection routines automatically bring up the scrolling EEG data viewer window, as shown in the screenshot below. You should scroll through the data to make sure that the parameters you entered have actually caused epochs containing artifacts to be detected without accidentally marking a large number of epochs without true artifacts. For example, you may need to increase the threshold if epochs without real artifacts are being marked for rejection, and you may need to decrease the threshold if epochs with clear artifacts are not being marked. Each epoch that has been marked is shown with a yellow background, and the channels within the epoch that exceeded the rejection threshold are drawn in red (see screenshot below). At present, artifact rejection is all-or-none; the entire epoch is discarded if an artifact is detected in one or more channels in that epoch.



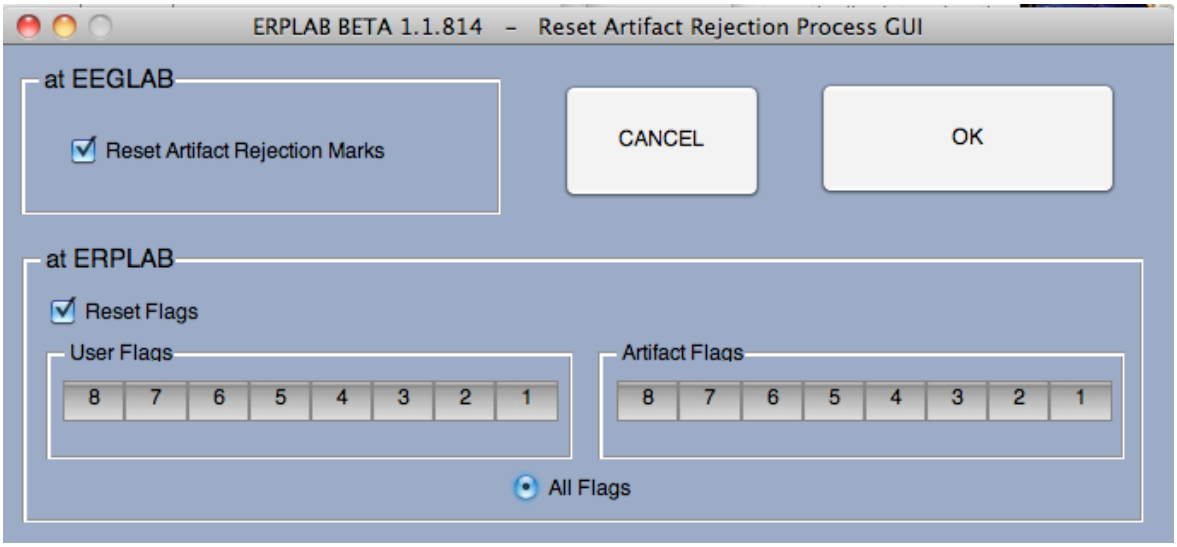
The window for saving the newly created dataset is also brought up at this point. The idea is that you will scan through the dataset, and either save it (if everything looks OK) or click on **CANCEL** (if you want to change the parameters and try again). Once you have finished looking at the data, click

Ok in the window for saving the dataset if you are satisfied with the artifacts that were detected (it will be named "S1_EEG_elist_be_ar").

Note that the scrolling EEG display window ordinarily allows you to click on an epoch that has not been marked to manually mark it or to click on an epoch that has already been marked to unmark it. However, this ability is disabled when you are looking at the initial results of an artifact detection procedure, because the goal at this moment is to determine whether the automated procedure worked correctly. Once you are satisfied and have saved the dataset, you can open it again in the scrolling EEG display window, and then you can manually reject and unreject epochs.

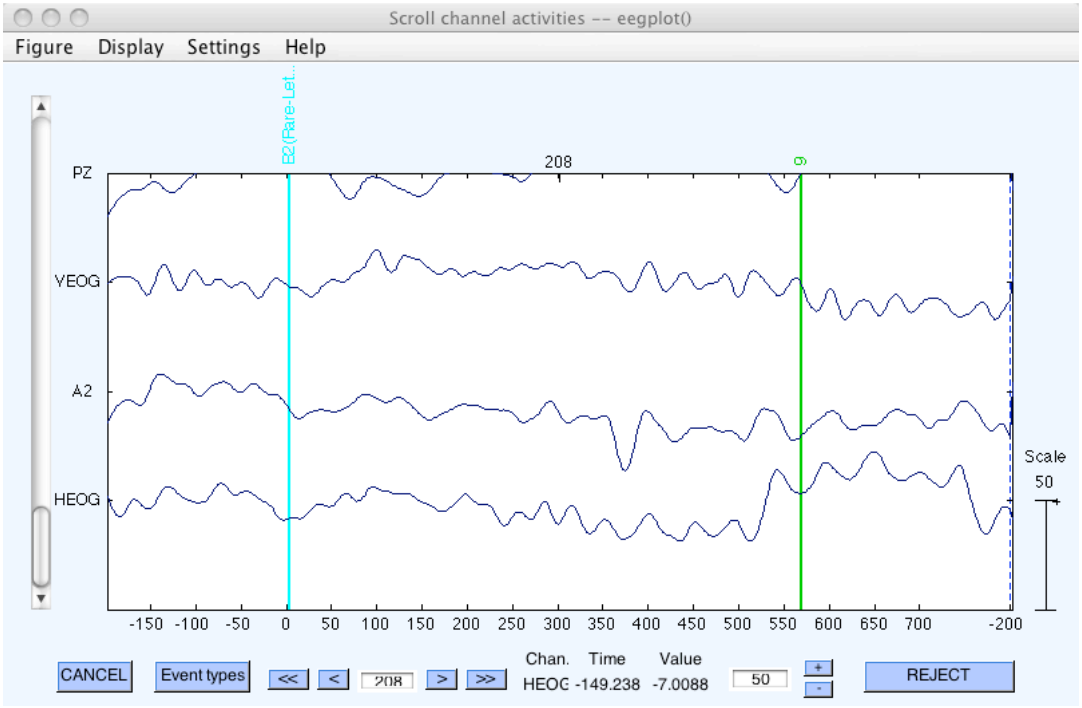
If, at some later time, you want to see how many trials have been marked for rejection in a given dataset, you can select **ERPLAB > Artifact Detection > EEG Artifact Detection Summary Table**. This will print a table of the number of trials marked for rejection in each bin, along with the number of occurrences of each flag (which ordinarily corresponds to different types of artifacts). You can also select **ERPLAB > Artifact Detection > EEG Artifact Detection Summary Plot**, which will show the distribution of artifacts across channels. You can also get this information after averaging with **ERPLAB > Artifact Detection > ERP Artifact Detection Summary Table**.

Imagine that you want to do artifact rejection twice, once on the VEOG channel with a relatively low threshold (e.g., 75 μV) and short moving window length (e.g., 200 ms) to detect blinks, and once on all the other channels with a relatively high threshold (e.g., 200 μV) and long moving window length (e.g., 1000 ms) to detect other kinds of artifacts (e.g., EMG bursts, skin potentials, etc.). You can simply run the artifact detection routine once with one set of parameters, save the dataset, and then run the artifact detection routine on this new dataset with a new set of parameters. Any artifacts marked in the dataset prior to artifact detection will remain marked when artifact rejection is performed on that dataset, and any additional artifacts will be marked in the new dataset. If you wish to unmark everything and start over, you can either clear the datasets and go back to an unmarked dataset or select **ERPLAB > Artifact Detection > Clear Artifact Detection Marks on EEG** which then displays the following figure.



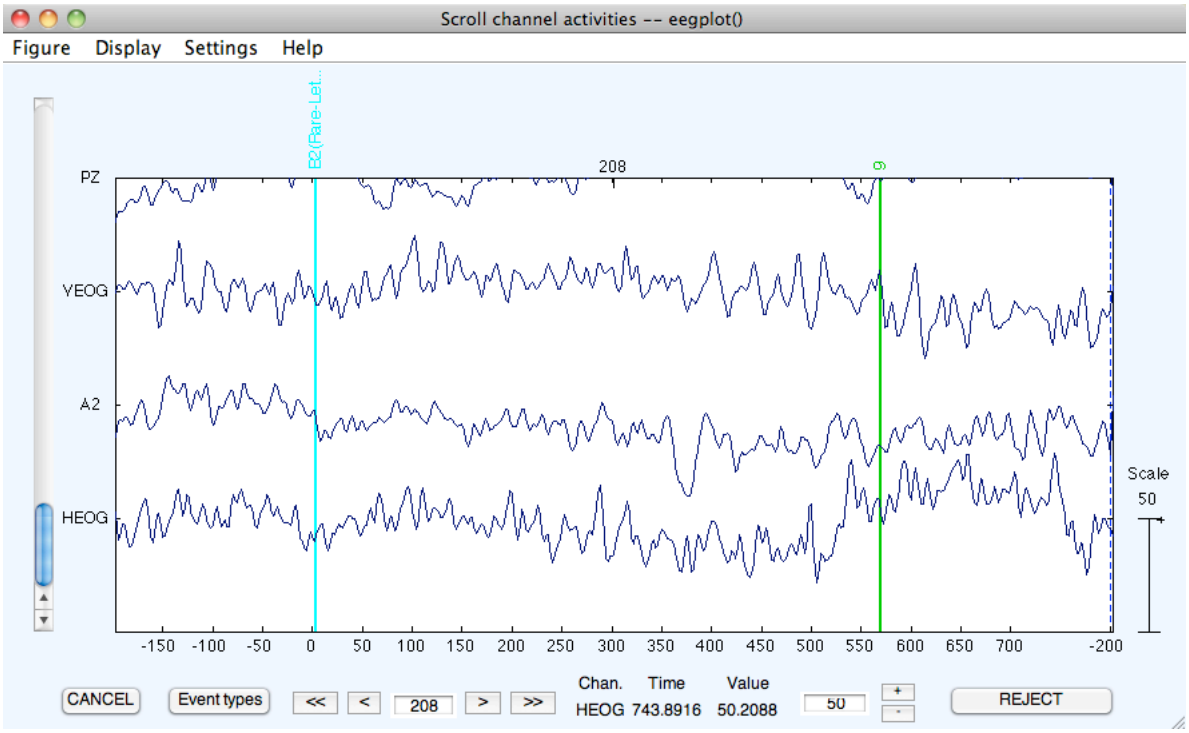
To reset the artifact detection marks, make sure **Reset Artifact Rejection Marks**, **Reset Flags**, and **All flags** are selected, then click **OK**. After you click **OK**, a window pops up prompting you to name the new dataset.

To see how this works, let's take the dataset that you've already marked with rejections using the **Moving window peak-to-peak threshold test** and apply another test, the **Step-like artifacts test**. In this example, we will attempt to detect horizontal eye movements in the HEOG channel (channel 16). Saccadic eye movements have a distinctive shape, because the eyes are steady at one location for a period of time, then jump to another location, stay steady for a period of time, and then jump to yet another location, so the HEOG signal is relatively flat, suddenly jumps to a new level, stays flat for a period of time, and then jumps to a new level. In many cases, the eyes are on the fixation point, jump away for a few hundred milliseconds, and then jump back to fixation. When this happens, the HEOG signal has a boxcar shape, as in the screenshot shown below.

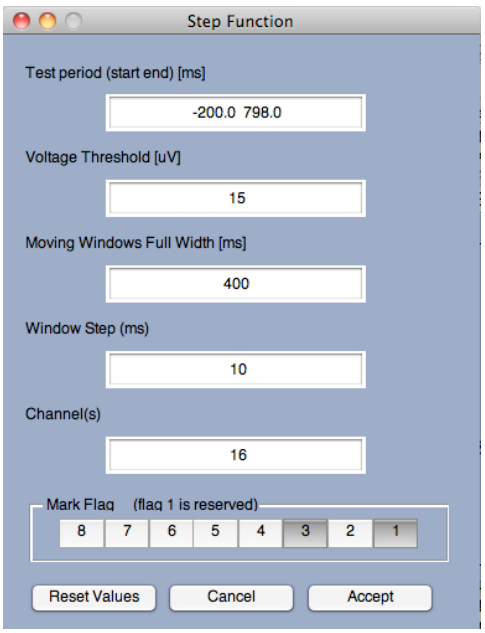


The current data file has quite a bit of high-frequency noise that makes it difficult to see this small artifact (which was at epoch #208). Thus, the data shown in the screenshot above were first filtered with a 30-Hz low-pass filter (see the section on filtering for more details). To get a close-up view of the artifact, the time window to display was set to 1 epoch and the number of channels to display was set to 3 (within the **Settings** menu within the plotting window).

Without the filter, the plot will look like the following screenshot:



The **Step-like artifacts** function is well suited to detecting this type of artifact (see the User's Manual for more details about this function). To use it, make sure that the dataset you created with the first artifact test is selected, and then select **ERPLAB > Artifact Detection > Step-like artifacts**. Enter the parameters as shown in the screenshot below, making sure that Flag 3 is selected (you should have selected Flag 4 when you did the peak-to-peak detection) and that you are detecting artifacts only in Channel 16. Note that the full width of the step function is set to 400 ms. This means that we are looking for a period of one voltage for >=200 ms immediately followed by a period of a different voltage for >=200 ms. This width was chosen because fixation periods almost always last for at least 200 ms.



After you click **Accept**, you will see in the command window that this routine found artifacts on 12.8% of trials. If you look at the scrolling data window that pops up, you will see that some of the epochs with these artifacts also had blink artifacts that were detected with the peak-to-peak amplitude routine that was previously run with these data (e.g., epoch 44 indicates a rejection of the HEOG channel by drawing this channel in red, and the data from several other channels are also drawn in red from the peak-to-peak artifact detection). However, other epochs were rejected solely because of the step function in the HEOG channel (e.g., epochs 208 and 209).

Information about the artifacts can also be found in the EventList. To see this, save the EventList from the current dataset in a text file by selecting **ERPLAB > EventList > Export EEG EventList to Text File**. Now open the text file in a text editor. You will see something like the following.

#item	bepoch	ecode	label	onset (sec)	diff (sec)	dura (msec)	b_flags (binary)	a_flags (binary)	enable	bin
157	79	22	Frequent_Digit	117.5840	0.3800	0.0	00000000	00001001	1	[1]
158	0	9	Corr_Resp	118.2980	0.7140	0.0	00000000	00000000	1	[ ]
159	80	22	Frequent_Digit	118.7840	0.4860	0.0	00000000	00000101	1	[1]
160	0	9	Corr_Resp	119.7060	0.9220	0.0	00000000	00000000	1	[ ]
161	81	12	Rare_Letter	151.0400	31.3340	0.0	00000000	00000000	1	[2]
162	0	9	Corr_Resp	151.4560	0.4160	0.0	00000000	00000000	1	[ ]
163	82	22	Frequent_Digit	152.2140	0.7580	0.0	00000000	00000000	1	[1]
164	0	9	Corr_Resp	152.6720	0.4580	0.0	00000000	00000000	1	[ ]
165	83	22	Frequent_Digit	153.4380	0.7660	0.0	00000000	00000000	1	[1]
166	0	9	Corr_Resp	153.8940	0.4560	0.0	00000000	00000000	1	[ ]
167	84	12	Rare_Letter	154.8120	0.9180	0.0	00000000	00001101	1	[2]

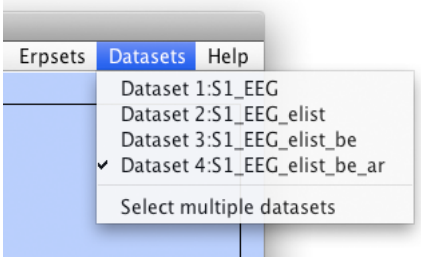
If you look at the column of artifact rejection flags (a_flags), you will see that every rejected trial has flag 1 set (the rightmost flag). Some of the events have flag 4 set from the peak-to-peak routine (e.g., item 157). Other events have flag 3 set from the step function routine (e.g., item 159), and other events have both flags 3 and 4 set (e.g., item 167), indicating that both types of artifacts were detected. This information can be useful if you want to perform behavioral analyses excluding trials with specific types of artifacts.

Note that the left column in this text file shows the original item number of the event, from the original continuous dataset. The second column shows the bin epoch (*bepoch*) number, which is the item number after epoching. These numbers will be different when some event codes are not used as time-locking events when the data are epoched (e.g., the response event codes in the current example). If you are trying to find an event in the continuous data with **Plot > Channel data (scroll)**, you should use the item number (first column). If you are trying to find an event in the epoched data, you should use the bepoch number (second column).

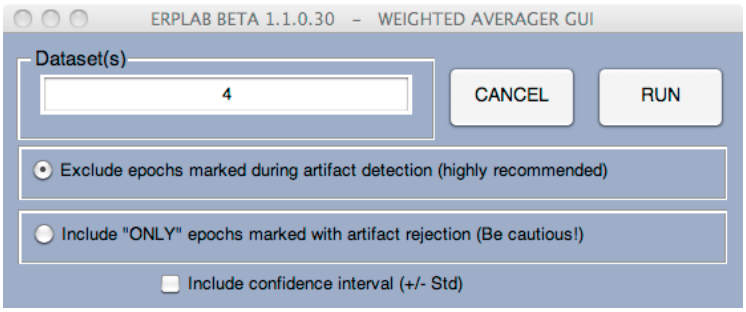
Note for programmers: The flags are shown in a binary format in the text file (to make it easier to see the individual flag values). However, they are stored in a single variable, and Matlab ordinarily uses decimal numbers to enter and print the value of a variable. For example: If you set flags 1 and 5 in an artifact detection tool, and you print out the value of the flag variable, you will see a value of 17.

9. Creating Averaged ERPs

We are now (finally!) ready to create a set of averaged ERP waveforms. First make sure that the correct dataset is active ("S1_EEG_elist_be_ar", which has been epoched and scanned for artifacts with the Moving Window peak-to-peak threshold artifact detection function). Your **Datasets** menu should look something like the screenshot below.



Now select **ERPLAB > Compute Averaged ERP**. This will bring up the very simple window shown below. You will normally average the data in the currently dataset, which is listed by default in the **Dataset(s)** field. If you want, you can specify multiple datasets and average across them as if all the epochs were stored in a single dataset. Here we will just average the epochs from one dataset, so click **RUN** to start the averaging process.


ERPLAB BETA 1.1.0.30 - WEIGHTED AVERAGER GUI
Dataset(s)
4
CANCEL
RUN
Exclude epochs marked during artifact detection (highly recommended)
Include "ONLY" epochs marked with artifact rejection (Be cautious!)
Include confidence interval (+/- Std)

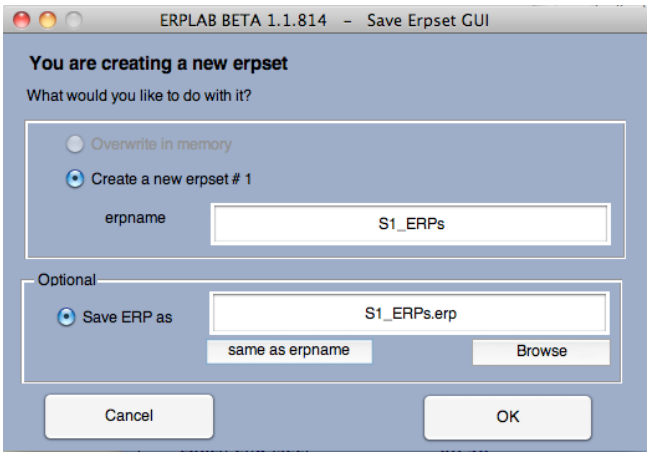
Hint: If you have a lot of channels or a high sampling rate (or both), you may find that the datasets are so large that Matlab becomes very slow (this is because only so much data can be stored in RAM, and Matlab may start storing some of the data in cache on your hard drive, which is very slow). In this case, you may wish to keep each trial block in a separate dataset and combine them into a single ERPset during averaging.

Normally you will want to exclude epochs marked as containing artifacts (marked with artifact flag 1 in the EventList), but you can include all trials by unchecking the "Exclude epochs marked during artifact rejection" box. It is also possible to include *only* the epochs that contain artifacts, excluding epochs without artifacts, by checking the "Include ONLY epochs marked during artifact detection" box.

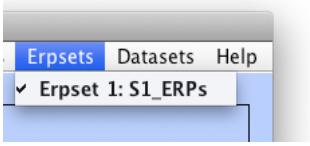
The averaging routine will print some useful information in the Matlab command window, as shown in the screenshot below. Specifically, it will show the percentage of trials that were excluded because of artifacts and because of invalid trials (e.g., epochs with **boundary** events within them; invalid trials should be rare). This information is provided both collapsed across bins and separately for each bin.

```
Averaging dataset #4...
averager.m : START

--------------------------------------------------------------------------------
The dataset S1_EEG_elist_be_ar has a 27.8 % of rejected trials
The dataset S1_EEG_elist_be_ar has a 0.0 % of invalid trials

TOTAL:
The dataset S1_EEG_elist_be_ar has a 27.8 % of  discarded trials

Summary per bin:
Bin 1 was created with a 29.3 % of rejected trials
Bin 1 was created with a 0.0 % of invalid trials
Bin 2 was created with a 21.9 % of rejected trials
Bin 2 was created with a 0.0 % of invalid trials
--------------------------------------------------------------------------------
averager.m : END
Done.
```

When the averaging routine is finished, it will display the window shown below that allows you to name and save the ERPset that was created. This window is shown every time an ERPLAB function modifies or creates an ERPset. If one or more ERPsets were already loaded, you would have the option of overwriting the previously active ERPset or creating a new ERPset (you will usually want to create a new one – ERPsets are usually much smaller than datasets, and you will not usually run out of RAM because of ERPsets). In this case, however, no other ERPsets are present, so you will be creating a new ERPset. You should name the ERPset "S1_ERPs". This is the name that will be shown in the ERPsets menu. You will usually want to save the results of averaging in a file on disk, so select the **Save ERP as** button and give the file the same name as the ERPset name. You can do this easily by clicking the **same as erpname** button. The file will be saved in Matlab's current directory unless you either specify an alternative path or use the **Browse** button. You can click the **OK** button once you have set up the window as shown in the screenshot below.



You can now see the newly created ERPset in the **ERPsets** menu, as shown in the screenshot below.

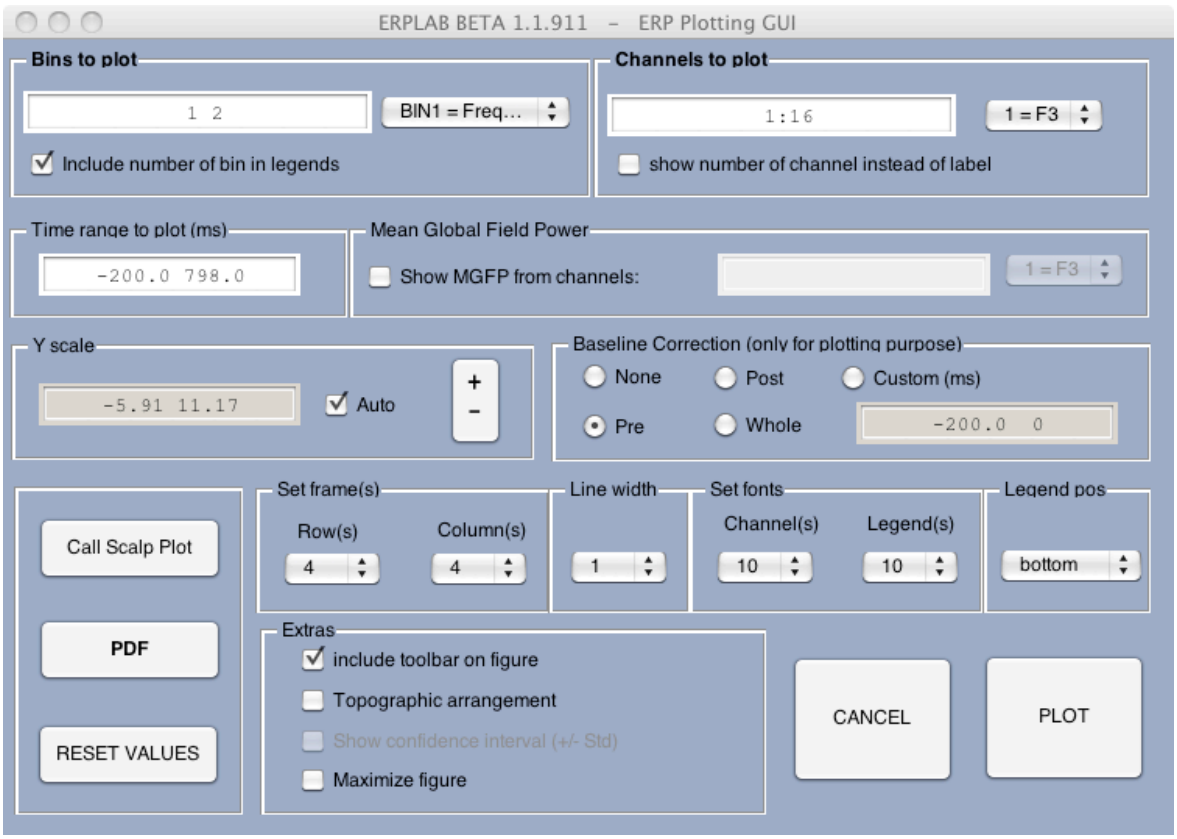


10. Plotting Averaged ERP Waveforms

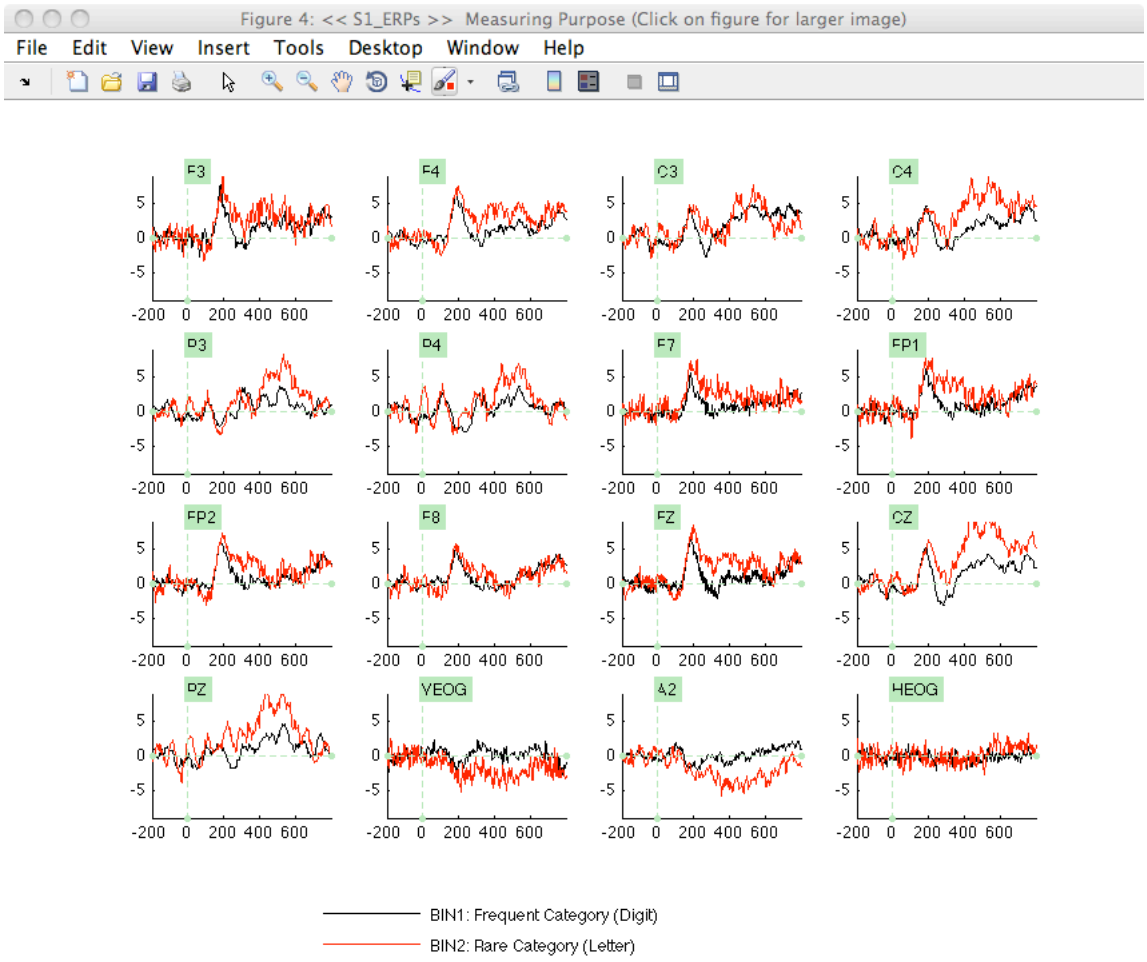
Now that you have created an ERPset, you can look at its contents by selecting **ERPLAB > Plot ERP Waveforms**, which brings up the window shown in the next screenshot. The top half of the

window allows you to specify what information to plot (bins, channels, etc.), and the bottom half allows you to control the details of the plotting (vertical scale, line width, font size, etc.). The window will remember the settings from the last time you used it, but you can revert to the default settings by clicking the **RESET VALUES** button (e.g., when you want to plot a new ERPset that has a different epoch length, set of channels, etc.).

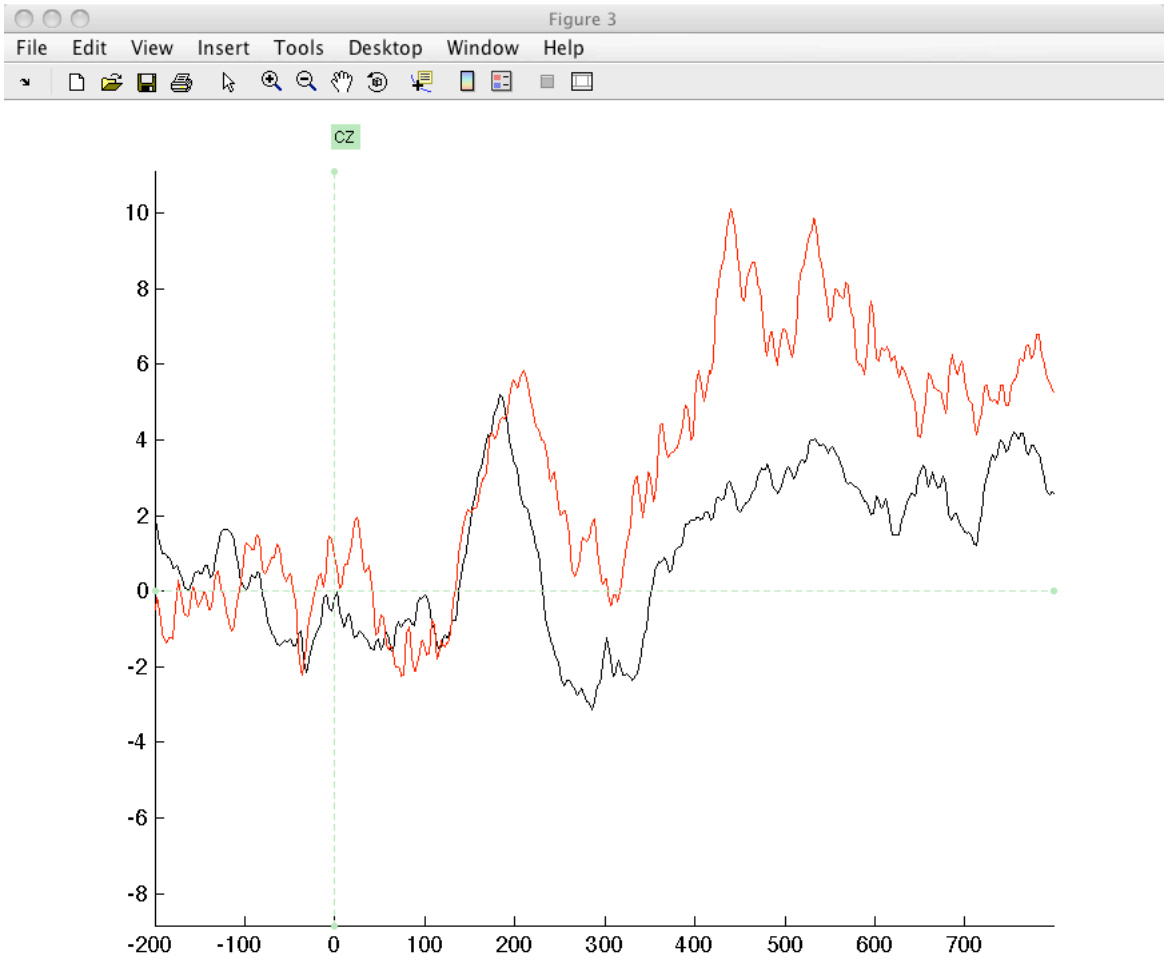
Hint: Do not try to select a new ERPset while the plotting window is open. The plotting window only knows about the ERPset that was active when the window was first opened. We will eventually change this window so that you can select among ERPsets from the plotting GUI.



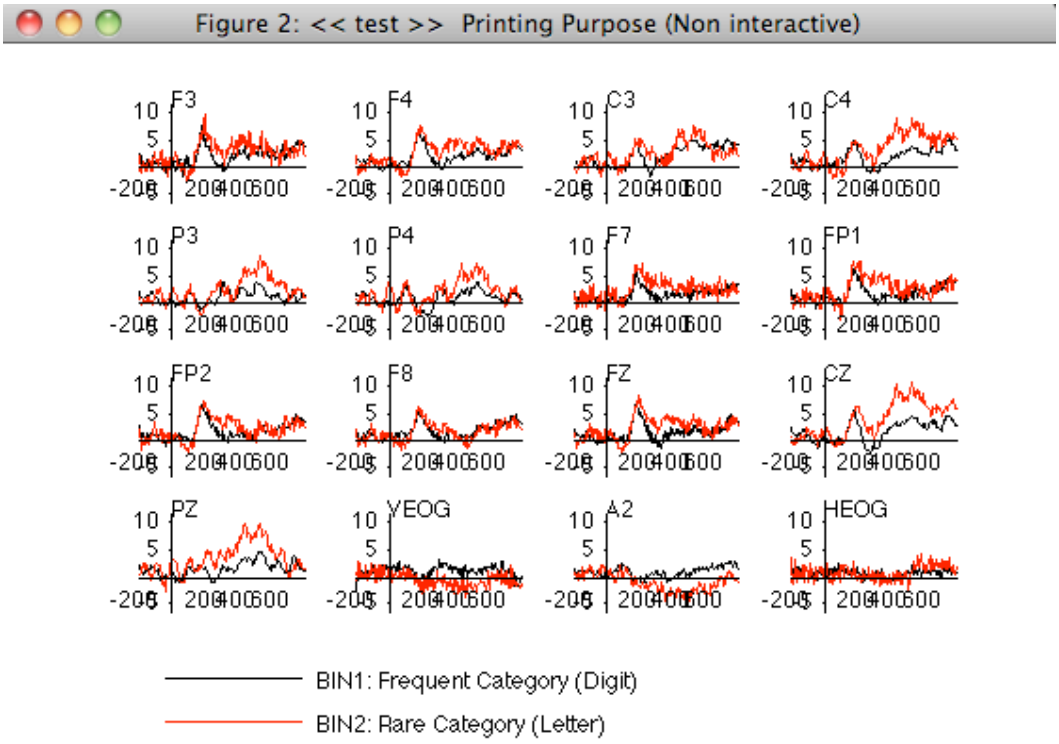
Click the **PLOT** button with the default values, and you should see a window pop up like the one shown below. At present, ERPLAB will create a separate panel for each scalp site and overlay the selected channels. In a future version, we will make this more flexible so that you can plot multiple ERPsets and choose which waveforms are overlaid and which waveforms are shown in different panels. For now, you can save the waveforms in a PDF file (by clicking the **PDF** button), import them into a graphics program such as Adobe Illustrator, and then edit them. We recommend that you maximize the figure window before clicking the PDF button. Alternatively, you can export the waveforms as text files using the **ERPLAB > Export ERP to Text** routines and then read them into a program such as Excel or Deltagraph for plotting.



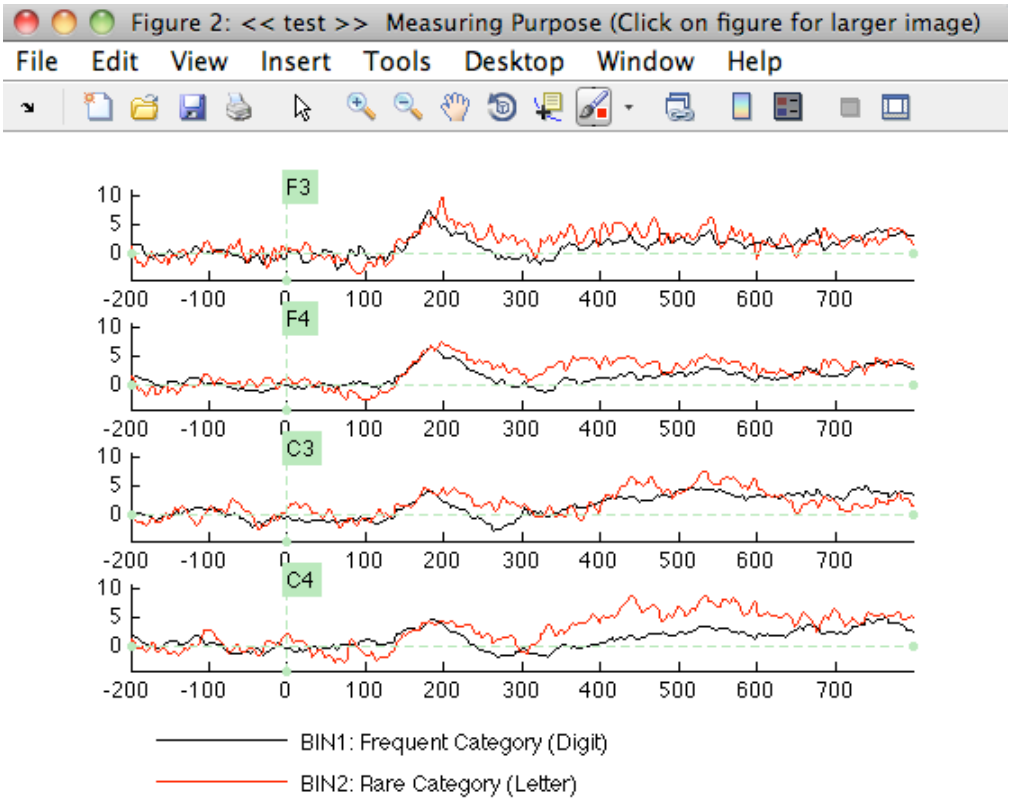
Note that, when **Include Toolbar** is checked in the plotting GUI, ERPLAB uses a standard Matlab plotting routine that provides a set of menus and a toolbar in the plotting window. Play around with it and you should be able to figure out how it works. Perhaps the most useful tool is the measurement tool (the icon with the small plus mark), which allows you to click on the waveform and see the X and Y values (time and voltage) at that point on the waveform. In addition, this mode allows you to click on one of the panels (e.g., the panel for the Cz electrode site), which will pop up a new window showing an expanded version of that panel, as shown in the screenshot below.



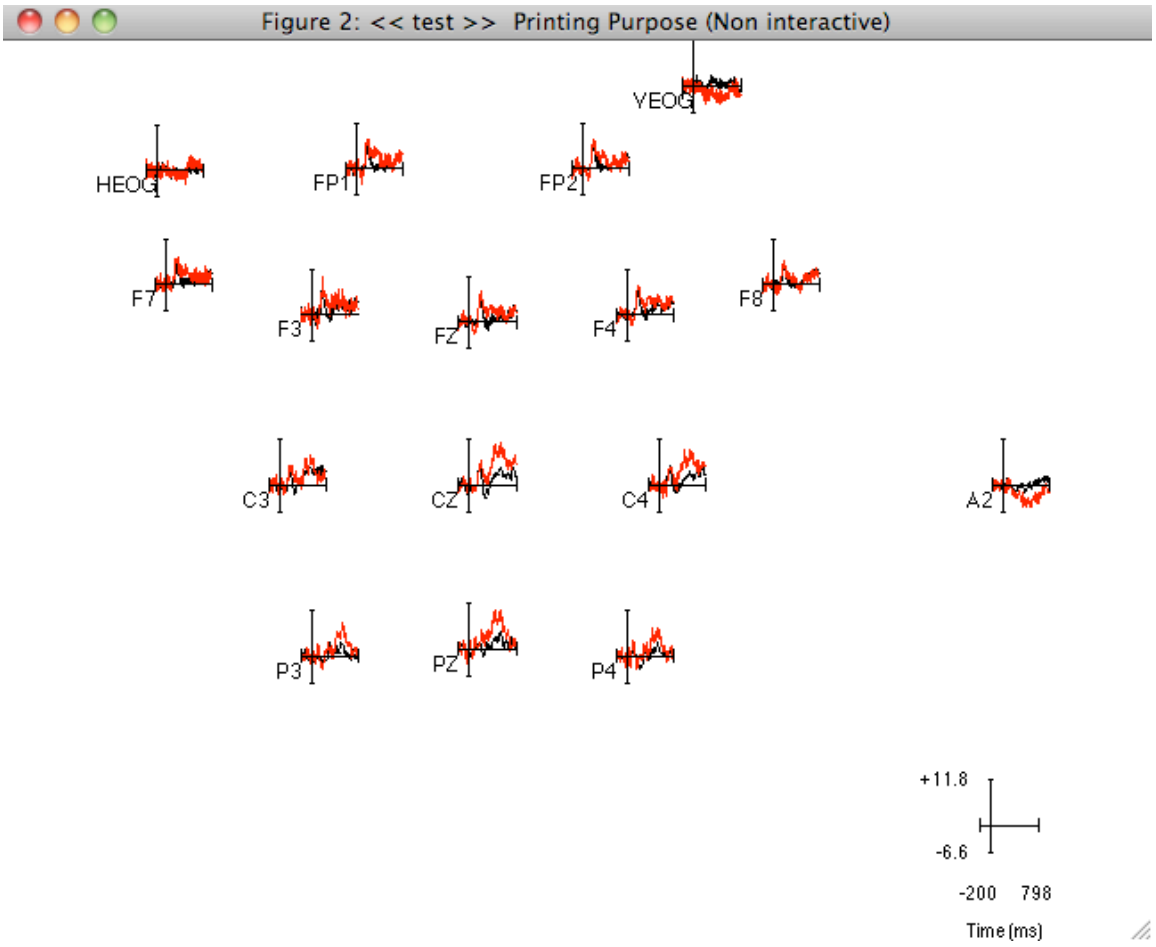
If **Include toolbar** is not checked, then these features are not present. In this mode, however, the X axis is plotted at zero microvolts rather than at the bottom of each panel, which is the appropriate way to plot the waveforms for publications and formal presentations. Try plotting the data again with **Include toolbar** unchecked; the result should look like the screenshot below.



There are many things you can do with the Plotting Waveform GUI. For example, you could plot a single column with multiple channels. For example, set **Channels to plot** to **1:4**, set **Row(s)** to **4**, and set **Column(s)** to **1**. You will see something like this:



Also, try checking the **Topographic** checkbox to create something like the following screenshot. There are **w** (width) and **h** (height) parameters that you can used to adjust the width and height of each waveform within the figure.



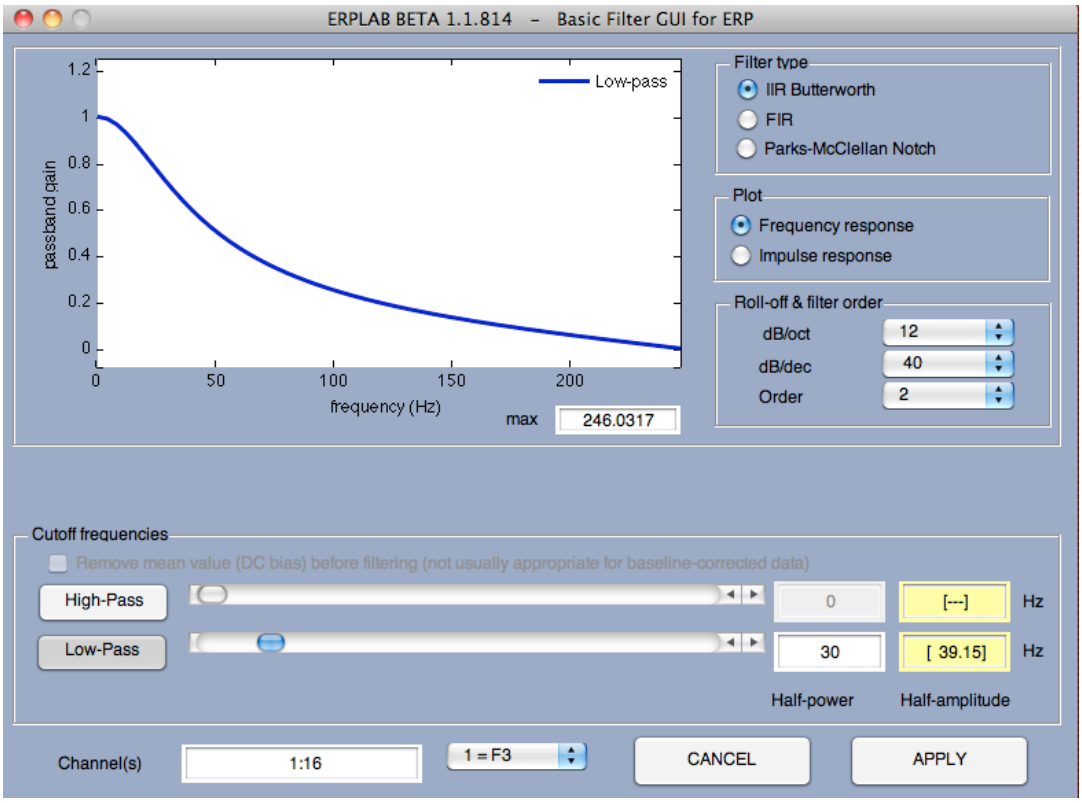
11. Filtering EEG and ERPs

ERPLAB contains a set of filters that can be applied to continuous EEG, to segmented EEG, and to averaged ERPs. Separate routines are used for filtering EEG (**ERPLAB > Filter & Frequency Tools > Filters for EEG data**) and for filtering ERPs (**ERPLAB > Filter & Frequency Tools > Filters for ERP data**). The routines are nearly identical, except for a few options. We will start by discussing the filtering of averaged ERPs. Make sure that "S1_ERPs" is selected in the **ERPsets** menu, and then select **ERPLAB > Filter & Frequency Tools > for ERP data**. You will see a window similar to the screenshot below. This window allows you to set various properties of the filter. First, you can determine whether to apply a low-pass filter (filtering out high frequencies) and/or a high-pass filter (filtering out low frequencies). For each of these, you can select the 50% amplitude cutoff point by typing a number in the corresponding text field or by adjusting the sliders. You can also adjust the slope of the filter's roll-off in the **Roll-off & filter order** panel in units of decibels per octave (**dB/oct**) or in units decibels per decade (**dB/dec**) or in terms of the filter's order (if you change one, the others will automatically change).

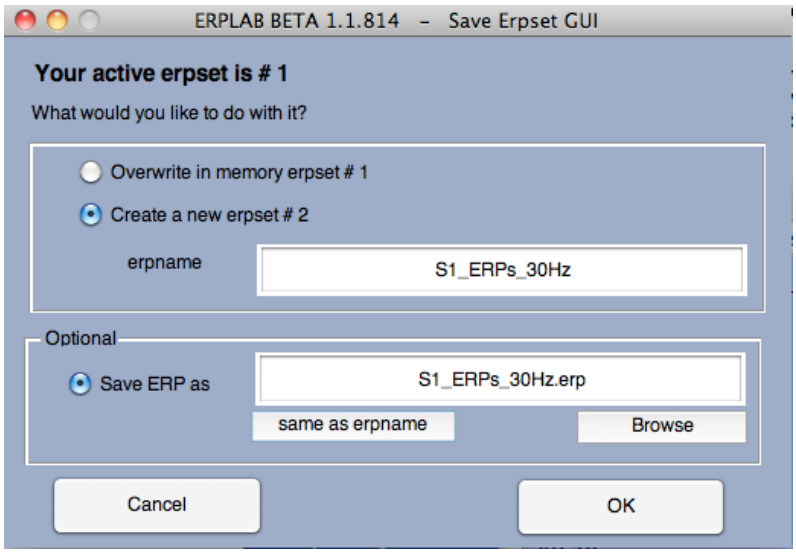
You can see the frequency response function of the resulting filter (the extent to which each frequency is passed by the filter) in the graph at the top of the window (assuming that the **Plot** panel is set to **Frequency response**). Try playing around with the various filter parameters to see how it changes the frequency response function.

Three general classes of filters have been implemented. First, **IIR Butterworth** is a digital implementation of the classic Butterworth filter. The output of this filter feeds back into its input, making it an *infinite impulse response* filter. It filters the data in both directions, avoiding any phase shift. Second, **FIR** is a *finite impulse response* filter that uses a sinc impulse response function. This kind of filter can be very efficient but can produce artificial oscillations in the filtered data under some conditions. We will eventually implement Gaussian impulse response functions but have not done so yet. However, the sinc filter approximates a Gaussian when a relatively low filter order is used. The third option is **Parks McClellan Notch**, which provides a notch filter (which attenuates one narrow range of frequencies and passes everything else).

Filters can dramatically distort your data, creating artificial peaks and oscillations (see Chapter 5 in *An Introduction to the Event-Related Potential Technique*). To avoid these distortions, we recommend that you choose a high-pass cutoff of 0.1 Hz or lower and a low-pass cutoff of 20 Hz or higher unless you really know what you're doing. We also recommend a roll-off slope of 12 dB/octave. For an example of how severe filtering can lead to incorrect conclusions, see Yeung et al., 2007, *Psychophysiology*, 44, 39-49.



As an example, set up the filter as shown in the screenshot above, with no high-pass filter, a low-pass cutoff at 30 Hz, a roll-of of 12 dB/octave, a Butterworth filter type, and all 16 channels being filtered. Click **APPLY**. After the filtered data have been created, the standard window for saving new ERPsets will appear, and you should name the new ERPset "S1_ERPs_30Hz" (you can save the ERPset to disk if desired, but this is not necessary). This will now be the active ERPset. If you plot it with **ERPLAB > Plot ERP Waveforms**, the result will look like the following screenshots (you should also plot the unfiltered data for comparison).


ERPLAB BETA 1.1.814 - Save Erpset GUI
Your active erpset is # 1
What would you like to do with it?
Overwrite in memory erpset # 1
Create a new erpset # 2
erpname
S1_ERPs_30Hz
Optional
Save ERP as
S1_ERPs_30Hz.erp
same as erpname
Browse
Cancel
OK

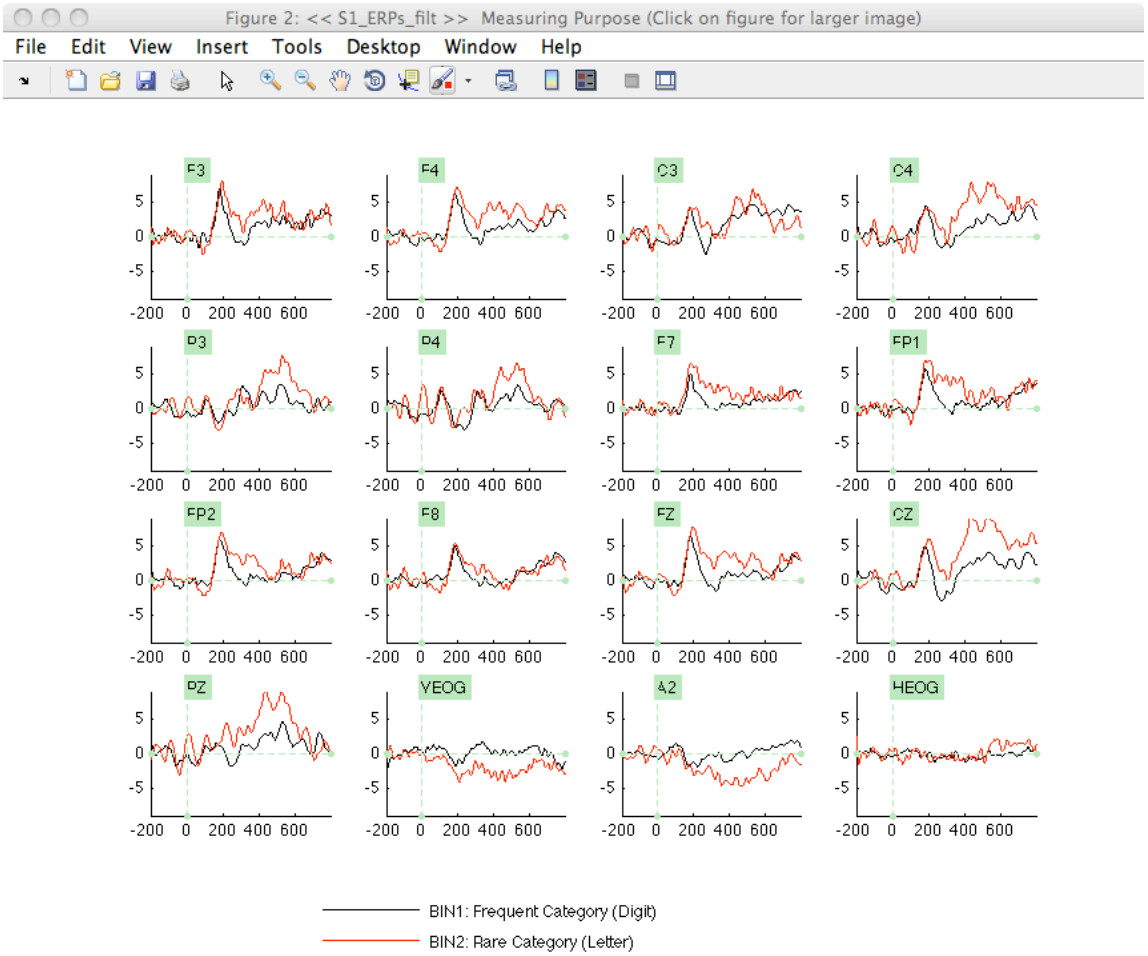
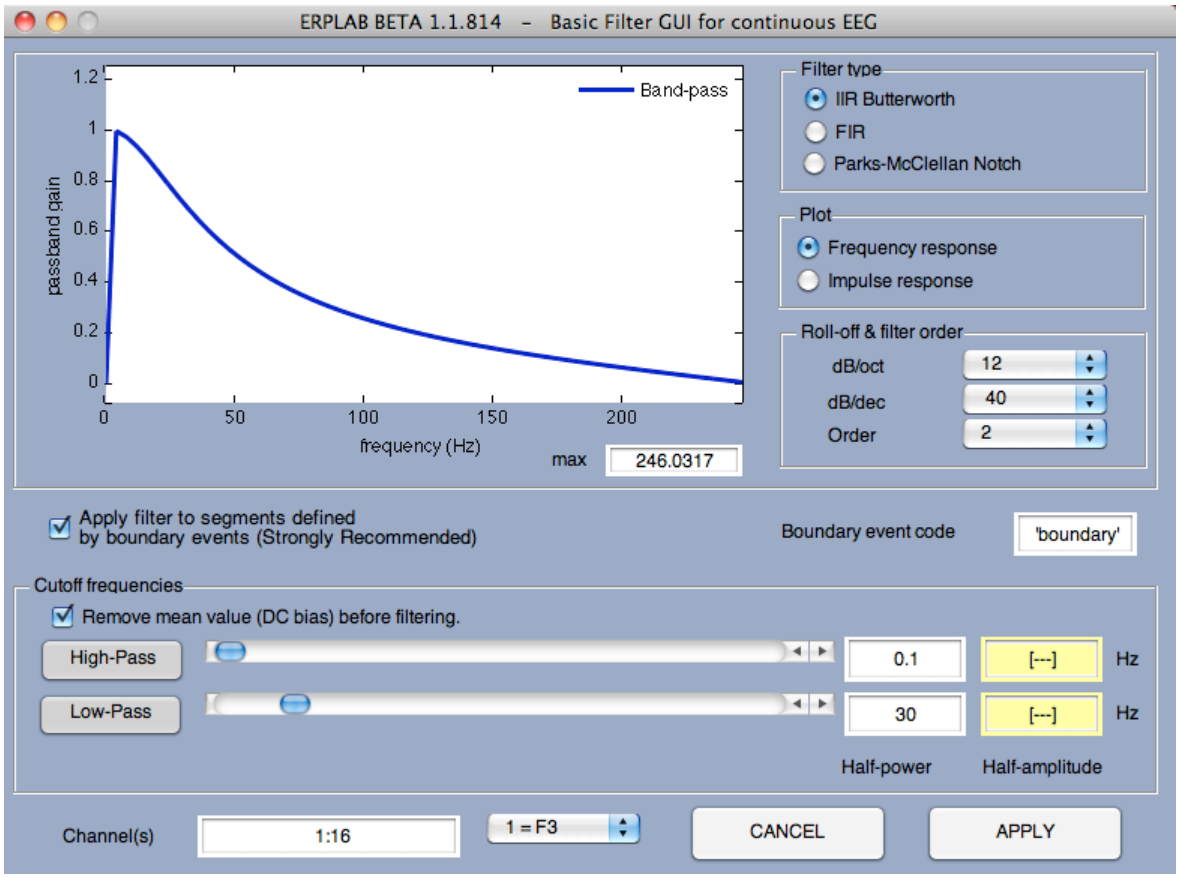
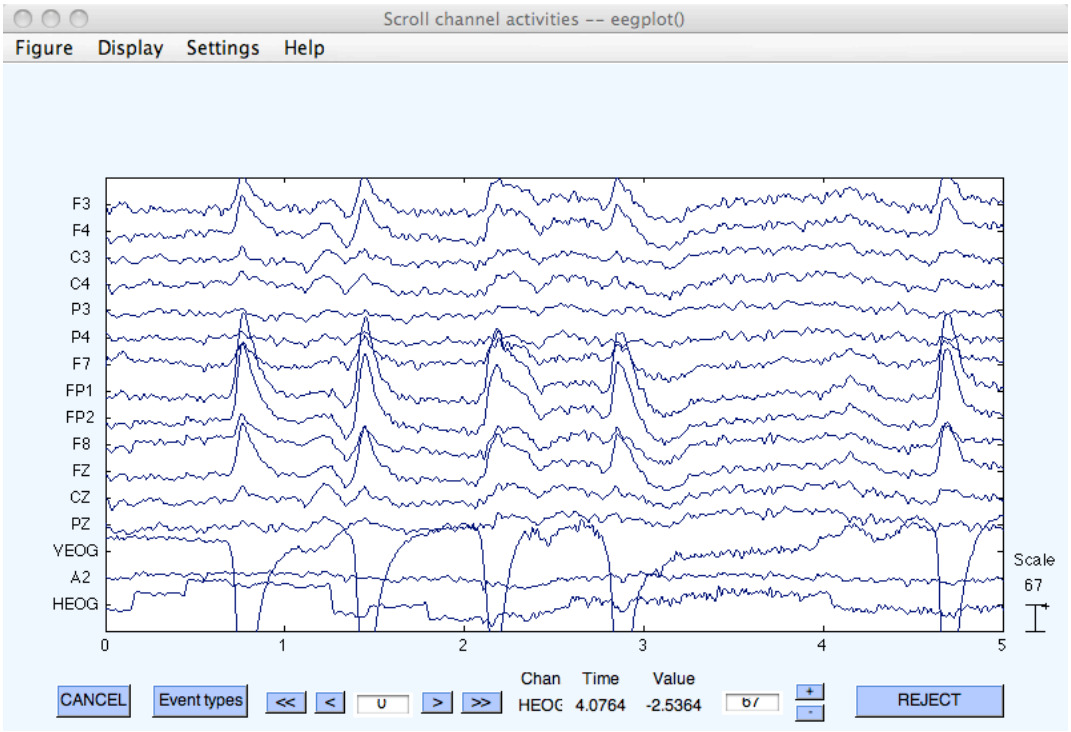


Figure 2: << S1_ERPs_filt >> Measuring Purpose (Click on figure for larger image)
File Edit View Insert Tools Desktop Window Help
F3
F4
C3
C4
P3
P4
F7
FP1
FP2
F8
FZ
CZ
PZ
VEOG
A2
HEOG
BIN1: Frequent Category (Digit)
BIN2: Rare Category (Letter)

Now let's try filtering continuous EEG data. Make "S1_Chan" the active Dataset (loading it with **File > Load existing dataset** if necessary). Select **ERPLAB > Filter & Frequency Tools > For EEG data**. Set the parameters as shown in the screenshot below, selecting a high-pass filter with a cutoff at 0.1 Hz and a low-pass filter with a cutoff at 30 Hz. You should check the boxes for applying the filter to segments defined by boundary events (which you should always check unless you *really* know what you're doing) and for removing the mean value (which you should always do with DC recordings and which won't hurt for non-DC recordings). The ERPLAB User's Manual provides more details.



Once you've set the parameters, click **APPLY**. The filtering will take a little while. Once it prints **Done** in the Matlab command window, use **Plot > Channel data (scroll)** to see the filtered data, which should look like the screenshot below.



Hint: In most cases, high-pass filters should be applied to the continuous EEG, not the epoched EEG or averaged ERPs. The reason is that, with typical cutoff values, a long period of continuous data is needed for these filters to work properly. Low-pass filters can be applied at any time, because they do not need a very long time period with typical settings. In most cases, however, it is best to apply low-pass filtering after averaging. Otherwise, you will need to back up to the EEG and repeat several steps if you later decide to change the filter settings. However, low-pass filtering can sometimes improve artifact detection, in which case you can apply the low-pass filter to either the continuous or epoched EEG prior to artifact detection.

12. Combining ERP Waveforms with Bin Operations

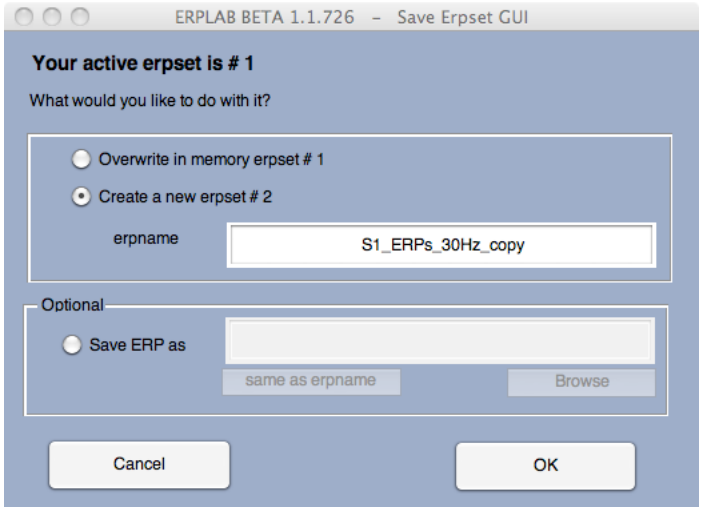
The **ERPLAB > ERP Bin Operations** routine allows you to create new bins by combining existing bins from the current ERPset. This is most often used to create difference waves or to average across bins.

Bin Operations has two modes of operation (see the **Mode** region at the bottom right in the screenshot below). In one mode, you use a set of equations to modify existing bins and add new bins within the current ERPset. In this mode, you can modify one bin and then use this modified bin to create or modify another bin (this is called *recursive updating*). In another mode, the current ERPset serves as the input to a set of equations that create a new set of bins in a new ERPset (this is called performing a set of *independent transformations*). We will begin with modifying an existing ERPset, which is the simplest way to simply add a few bins.

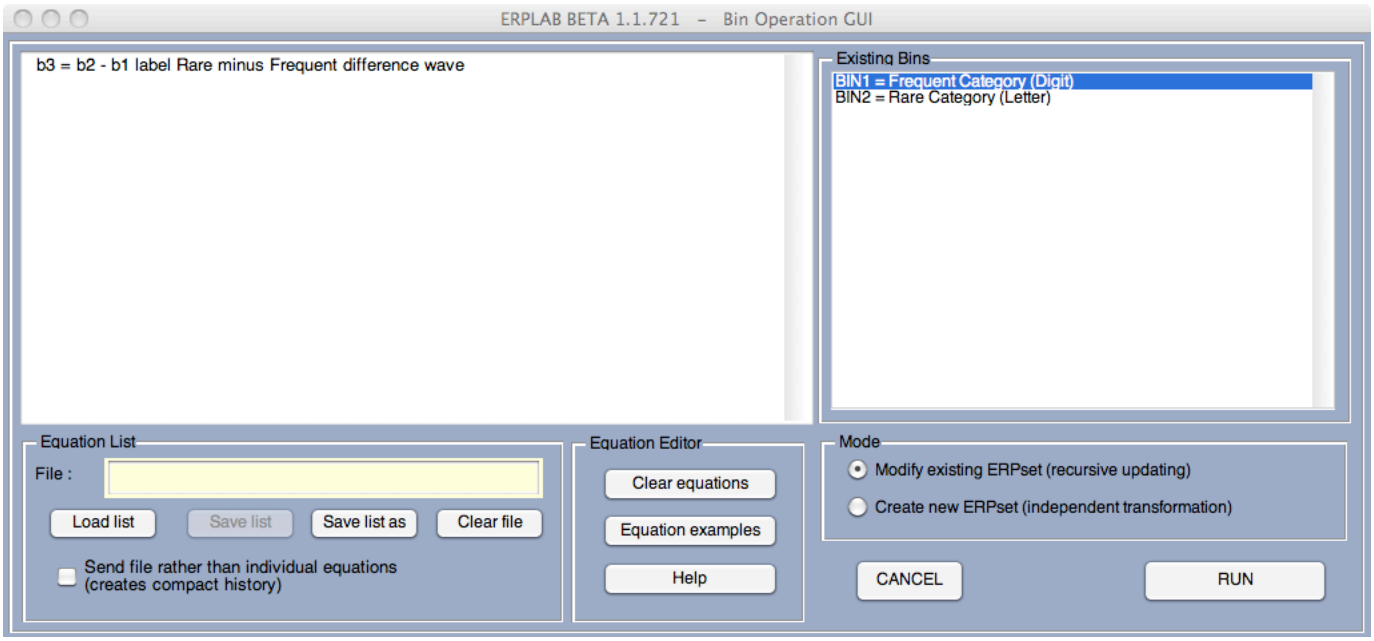
12.1. Modifying an existing ERPset (recursive updating)

As our first example, we will **ERPLAB > ERP Bin Operations** to make a difference wave, in which the ERP from the frequent stimuli is subtracted from the ERP from the rare stimuli. We will operate on the data in the ERPset you created earlier called "S1_ERPs_30Hz", so make sure it is selected in

the **ERPset** menu. It's usually a good idea to operate on a duplicate of an ERPset, so first select **ERPLAB > Duplicate or rename current ERPset** and then make a duplicate of it called "S1_ERPs_30Hz_copy", as shown in the screenshot below.



Now select **ERPLAB > ERP Bin Operations**. The window shown below will appear.

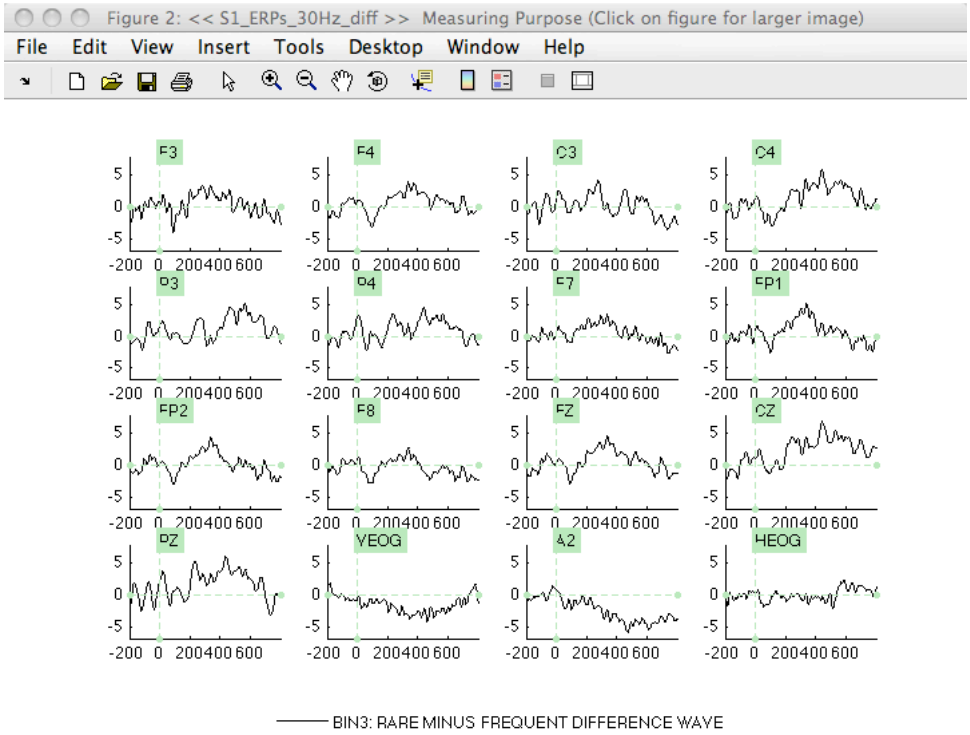


The right panel lists the existing bins in your ERPset, and the left panel shows a set of equations that you write to create new bins from the existing bins. In the example shown here, an equation is written to create a new bin 3 within the current ERPset that is equal to the existing bin 2 minus the existing bin 1 (the rare-minus-frequent difference wave). A label is also provided for the new bin. You can create as many new bins as you'd like in one step, defining one bin on each line of the left panel. You can save your bin equations in a text file (by clicking **Save list as**) and you can read in a previously saved set (by clicking **Load list**).

*Hint: You can click on the **Examples** button to see examples of how to define bins.*

Once you have defined the new bin as shown in the screenshot above, make sure that **Modify existing ERPset (recursive updating)** is selected, and then click **RUN** to create the new bin. In this mode of operation, the current ERPset will be directly modified. After you've updated the ERPset, you can save it to a file with **ERPLAB > Save Current ERPset**.

Once you've created the new bin, you can plot it with **ERPLAB > Plot ERP Waveforms**. If you select only bin 3, the result should look like the screenshot below.



In this example, we've simply added a new bin to the current ERPset. Note, however, that you can also update the existing bins in this mode. In addition, any changes you make in one equation can influence the results of the next equation. For example, after creating the difference wave in bin 3, you could have an equation that uses this difference wave to create a new bin 4.

12.2. Creating a new ERPset (independent transformations)

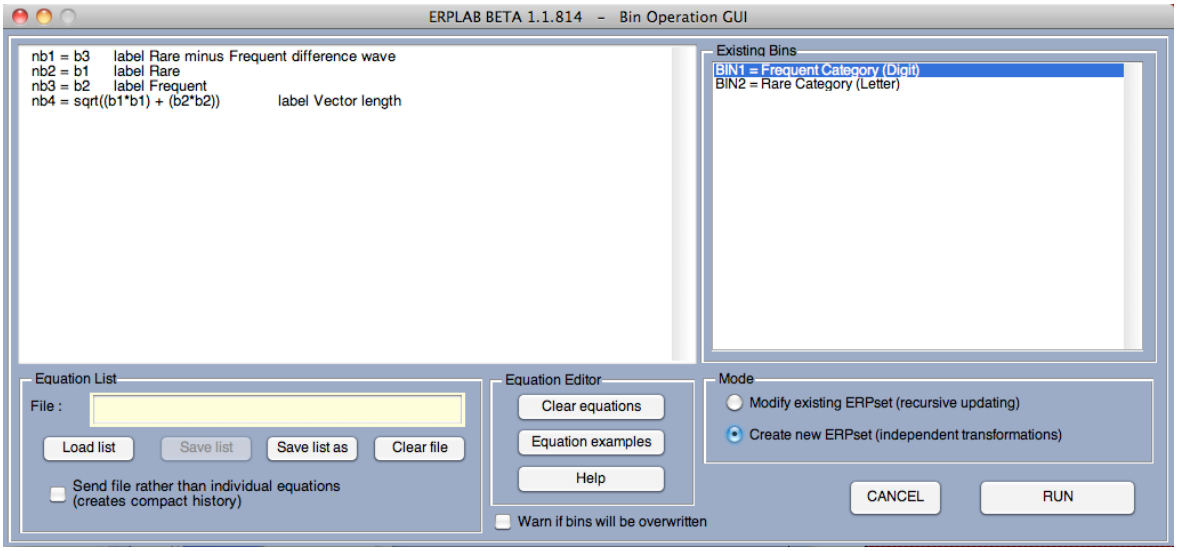
In this example, we will use **ERPLAB > ERP Bin Operations** to re-order the bins in the output from the previous step. We will also create an additional bin that shows how to implement a somewhat more complicated transformation. If you select **ERPLAB > ERP Bin Operations** after performing the previous step, you will see that the newly created bin (bin 3) is now listed in the right panel as an existing bin (see screenshot below). We are going to create a new ERPset in this example, so select **Create new ERPset (independent transformations)**. The equations in the left panel of the window will create a new ERPset in which the first 3 bins are re-ordered versions of the bins in the current ERPset and the 4th bin is the vector distance between the first two bins:

nb1 = b3 — This means that new bin 1 (nb1) in the new ERPset will be equal to the original bin 3 (b3) in the current ERPset)

nb2 = b1 — This means that new bin 2 (nb1) in the new ERPset will be equal to the original bin 1 (b1) in the current ERPset)

nb3 = b2 — This means that new bin 1 (nb1) in the new ERPset will be equal to the original bin 2 (b2) in the current ERPset)

nb4 = sqrt((b1*b1) + (b2*b2)) — This means that new bin 4 (nb4) is the Pythagorean distance between the original bins 1 and 2 (rare and frequent)



After you complete your equation list, you can choose to save it by clicking **Save list as**. You will prompted to save your ERPset. You can save this name as "S1_ERP_bo.txt" and just click **OK**.

Note that you can also recombine bins with different subsets of channels (e.g., to make contra-minus-ipsi difference waves to isolate the lateralized readiness potential or the N2pc wave). See the section on Bin Operations in the ERPLAB User's Manual for details. This can be done only in the **independent transformations** mode.

*Hint: There is no reason to first make a duplicate of the ERPset in this mode, because it always creates a new ERPset. In the **recursive updating** mode, however, creating a duplicate is useful (although not necessary) so that you can easily step backward and try again if you make a mistake or change your mind.*

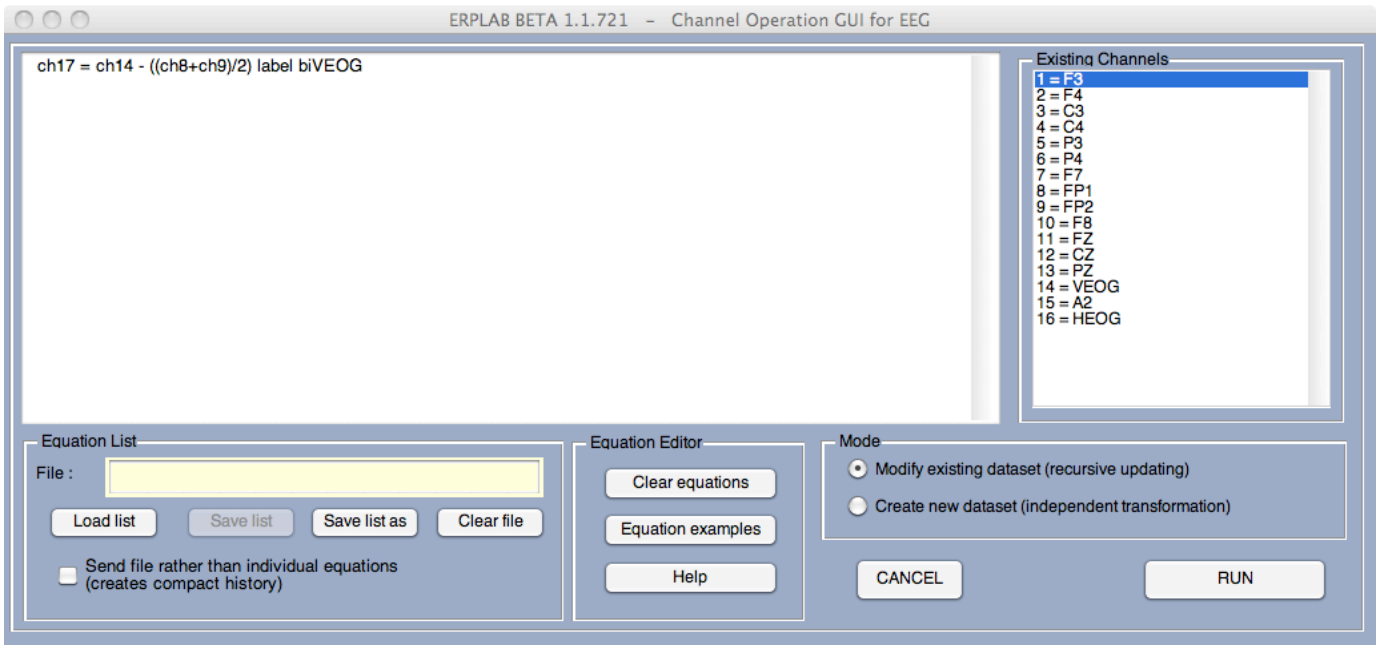
13. Creating and Modifying Channels with Channel Operations

We will now discuss how you can create and modify channels. The most common use for this is to re-reference all (or most) channels to a different reference electrode (or combination of electrodes). A second use is to create a bipolar channel that better isolates an artifact signal. This can be done either on the EEG in a dataset (**ERPLAB > EEG Channel Operations**) or on the ERPs in an ERPset (**ERPLAB > ERP Channel Operations**).

Like Bin Operations, Channel Operations has two modes of operation (see the **Mode** region at the bottom right in the screenshot below). In one mode, you use a set of equations to modify existing channels and add new channels within the current dataset or ERPset. In this mode, you can modify one channel and then use this modified channel to create or modify another channel (this is called *recursive updating*). In another mode, the current dataset or ERPset serves as the input to a set of equations that create a new set of channels in a new ERPset (this is called creating a set of *independent transformations*). Either mode can be used for most situations, but only the independent transformations mode can be used to re-order the channels, and the recursive updating mode can be more efficient for certain complicated transformations. We will show examples of both modes here.

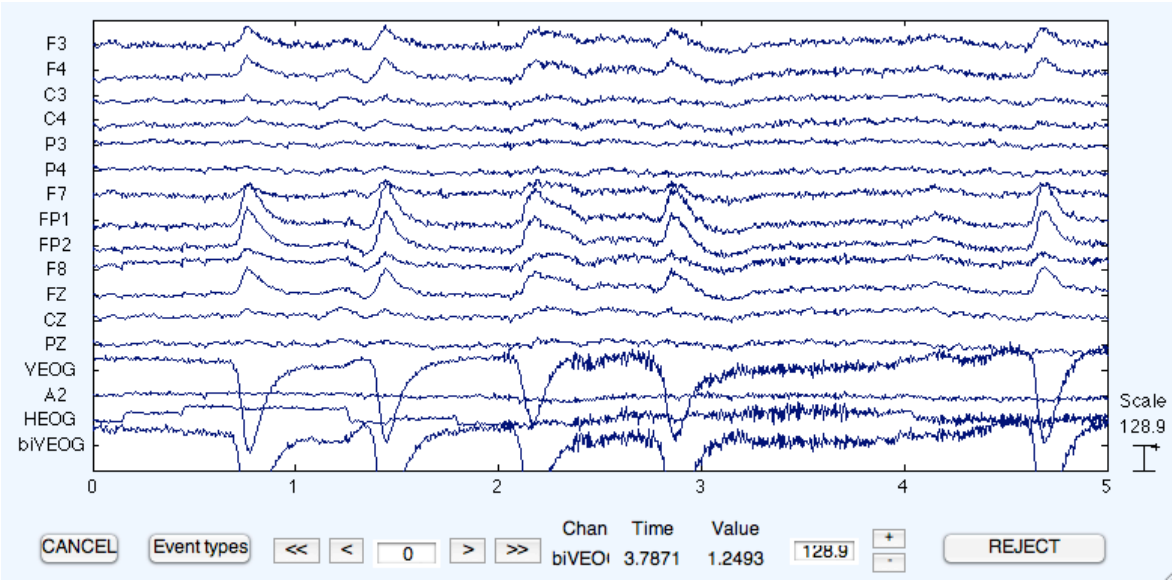
13.1. Creating a bipolar VEOG channel by updating the current dataset

In our first example, we will take the VEOG signal (which was recorded as the potential between an electrode under the left eye and a left mastoid reference electrode) and create a bipolar derivation representing the difference between the potential under the eye and the average of the Fp1 and Fp2 electrodes. To get started, make sure that the "S1_Chan" dataset is loaded and selected in the **Datasets** menu, and then select **ERPLAB > EEG Channel Operations**. This will bring up the window shown below. The right panel shows the channels in the current dataset, and the left panel shows a set of equations that will be used to define the bins in the newly created dataset. In this example, we will create a new channel (channel 17) that is the difference between the original unipolar VEOG channel (channel 14) and the average of Fp1 and Fp2 (channels 8 and 9), giving the new channel the label "biVEOG". Put the equation for this new channel into the panel on the left side of the window, as shown in the screenshot below. And because we are updating the current dataset, make sure that **Modify existing dataset (recursive updating)** is selected.



Once the window looks like the screenshot above, click **RUN**. You can also see in the EEGLAB menu that the Channels per frame have now been updated to 17. To see the result, select **Plot > Channel**

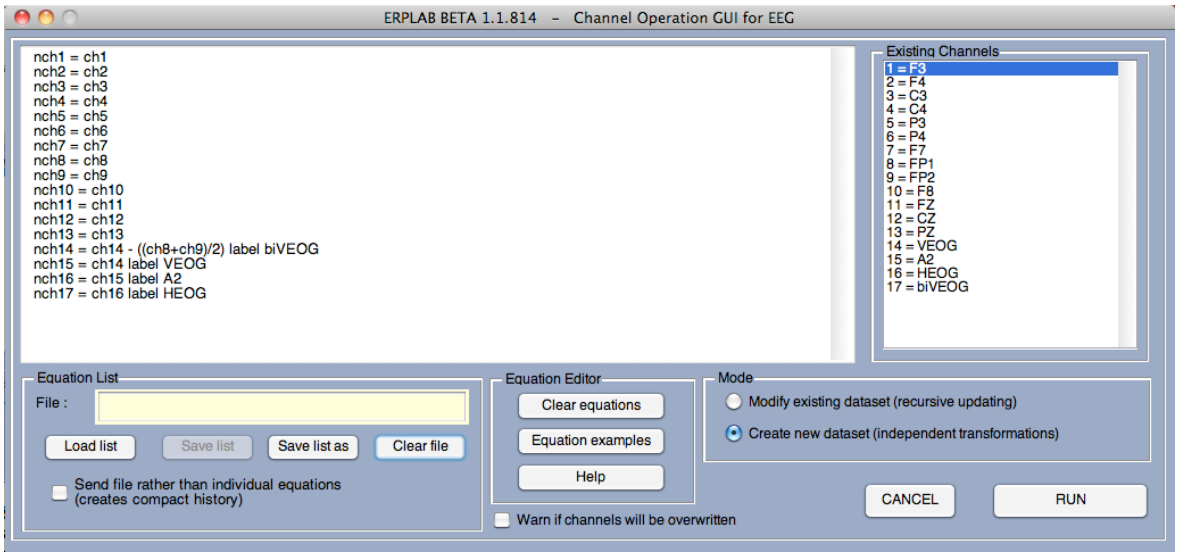
data (scroll). It should look like the screenshot below (you may need to increase the plotting scale).



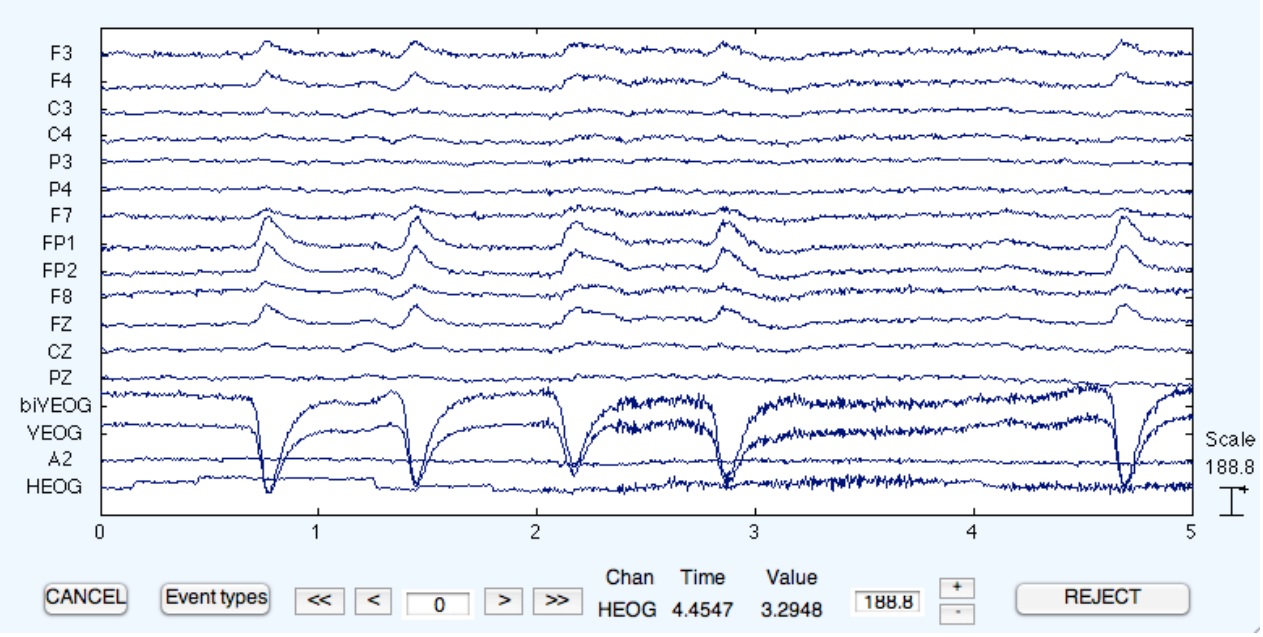
It's a bit difficult to see the new channel, because it is at the bottom of the window. It would be nice to insert this new channel prior to the original VEOG channel rather than at the end, but it is difficult to do this in recursive mode. The next example shows how to do this by creating a new dataset (independent transformations mode).

13.2. Inserting a bipolar VEOG channel by creating a new dataset

In this example, we will use the independent transformations mode to create the bipolar VEOG channel and place it in the middle of the set of channels rather than at the end. To get started, re-open the "S1_Chan" dataset (the existing S1_Chan dataset was modified in the previous example, and we want the original version). Now select **ERPLAB > EEG Channel Operations** and set it up as shown in the screenshot below. Because we are creating a new dataset, we need to copy any of the original channels (listed as ch1, ch2, ch3, …) that we want to keep to the new channels (listed as nch1, nch2, nch3, …). In this example, we first copy channels 1-13 from the current dataset to the new dataset, then add the bipolar VEOG as channel 14, and then copy channels 14-16 from the current dataset to channels 15-17 in the new dataset. Note that the left side of each equation must indicate that we are creating channels within a new dataset (e.g., nch1) rather than adding or modifying channels within an existing dataset (e.g., ch1).



This is a pretty long list of equations, and you wouldn't want to retype it every time you want to perform this transformation. To avoid retyping, you can save the list of channel equations to a text file (**Save list**) and reload it later (**Load list**). You can also use a regular text editor (such as the Matlab text editor) to create and modify these lists of equations, which can be convenient for complex transformations that contain many lines.

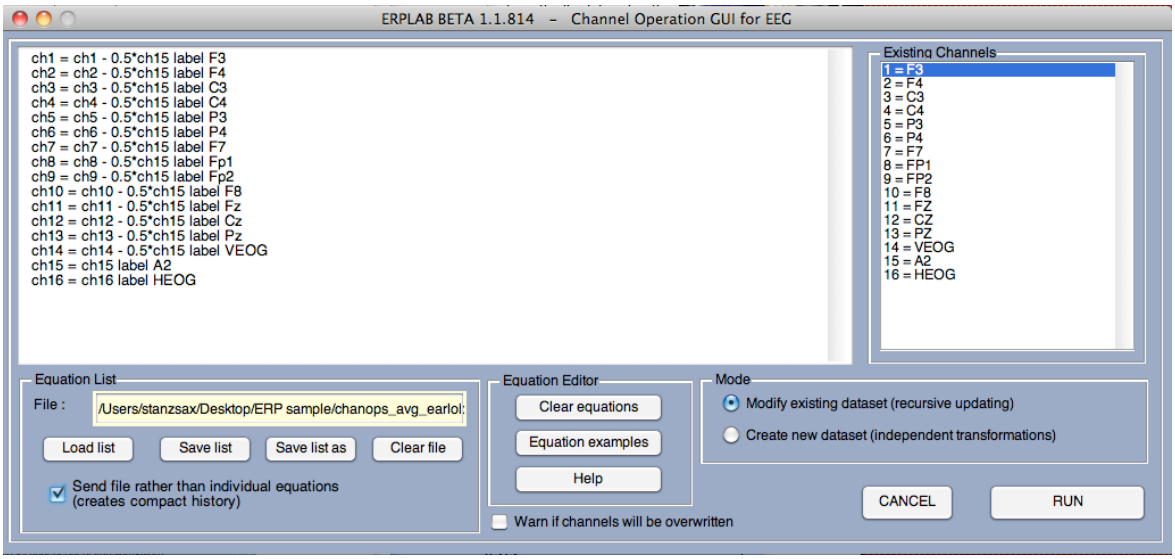


Once you have the window set up as shown in the screen shot above, click the **RUN** button. You will then be asked how to save the new dataset. Name it "S1_Chan_biVEOG" (you can also save it to the disk if you'd like). If you plot it using **Plot > Channel data (scroll)**, you will see something like the following screenshot (you may need to increase the plotting scale to make it look like this). Note that the blinks look somewhat larger in the new biVEOG channel than in the original VEOG channel.

13.3. Re-referencing to the average of the left and right earlobes or mastoids

You can also use Channel Operations to re-reference your data to a common, derived reference, such as the average of the left and right mastoids, the average of the left and right earlobes, or the average of all scalp sites. In this example, we will re-reference to the average of the left and right earlobes (the next example will show how to re-reference to the average of all electrodes). To begin, re-load the "S1_Chan" dataset. Each channel in this dataset, with the exception of HEOG, was originally referenced to the left earlobe (A1). To re-reference to the average of the left and right earlobes, we need to subtract 50% of the right earlobe (A2) signal from all of the sites (see pp. 107-108 in Luck, 2005). But we don't want to re-reference the A2 site or the HEOG site. This would be done using the equations shown in the screenshot below. In this example we are updating the existing dataset (recursive mode), but it would work exactly the same way in the independent transformations mode (except that the left part of each equation would start with nch1, nch2, etc. rather than ch1, ch2, etc.).

The set of equations is fairly long, so you may want to load it from the file **chanops_avg_earlobes.txt**. You can see that each channel is updated to equal its original value minus half of channel 15 (A2), with the exception of channels 15 and 16. These channels are simply equal to themselves. It isn't actually necessary to define these channels as being equal to themselves when updating an existing dataset (because the channels are already there), but it can be useful to include this as a reminder that these channels are present. When everything is ready, click **RUN**. This should update the current dataset. If you plot the data before and after re-referencing, you should be able to see small differences.

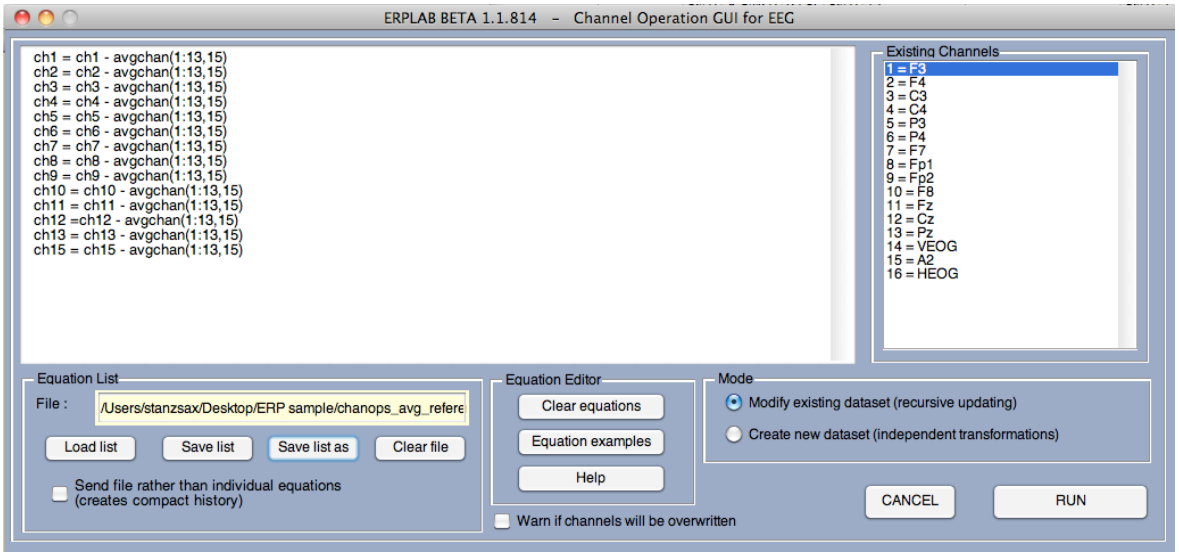

ERPLAB BETA 1.1.814 - Channel Operation GUI for EEG
ch1 = ch1 - 0.5*ch15 label F3
ch2 = ch2 - 0.5*ch15 label F4
ch3 = ch3 - 0.5*ch15 label C3
ch4 = ch4 - 0.5*ch15 label C4
ch5 = ch5 - 0.5*ch15 label P3
ch6 = ch6 - 0.5*ch15 label P4
ch7 = ch7 - 0.5*ch15 label F7
ch8 = ch8 - 0.5*ch15 label Fp1
ch9 = ch9 - 0.5*ch15 label Fp2
ch10 = ch10 - 0.5*ch15 label F8
ch11 = ch11 - 0.5*ch15 label Fz
ch12 = ch12 - 0.5*ch15 label Cz
ch13 = ch13 - 0.5*ch15 label Pz
ch14 = ch14 - 0.5*ch15 label VEOG
ch15 = ch15 label A2
ch16 = ch16 label HEOG
Existing Channels
1 = F3
2 = F4
3 = C3
4 = C4
5 = P3
6 = P4
7 = F7
8 = FP1
9 = FP2
10 = F8
11 = FZ
12 = CZ
13 = PZ
14 = VEOG
15 = A2
16 = HEOG
Equation List
File :
/Users/stanzsax/Desktop/ERP sample/chanops_avg_earlol:
Load list
Save list
Save list as
Clear file
Send file rather than individual equations (creates compact history)
Equation Editor
Clear equations
Equation examples
Help
Mode
Modify existing dataset (recursive updating)
Create new dataset (independent transformations)
CANCEL
RUN
Warn if channels will be overwritten

Note that you can apply Channel Operations on either the EEG data in a dataset (using **ERPLAB > EEG Channel Operations**) or the ERP data in an ERPset (using **ERPLAB > ERP Channel Operations**). In most cases, you will perform linear transformations of the data with these operations (i.e., new channels are created by adding together a set of scaled old channels). The order of operations does not matter for linear operations. Averaging is also a linear operation. Consequently, performing channel operations on the EEG and then averaging will give you the same results as averaging the EEG and then performing channel operations. However, artifact detection/rejection is a nonlinear process, so it will matter whether you perform channel

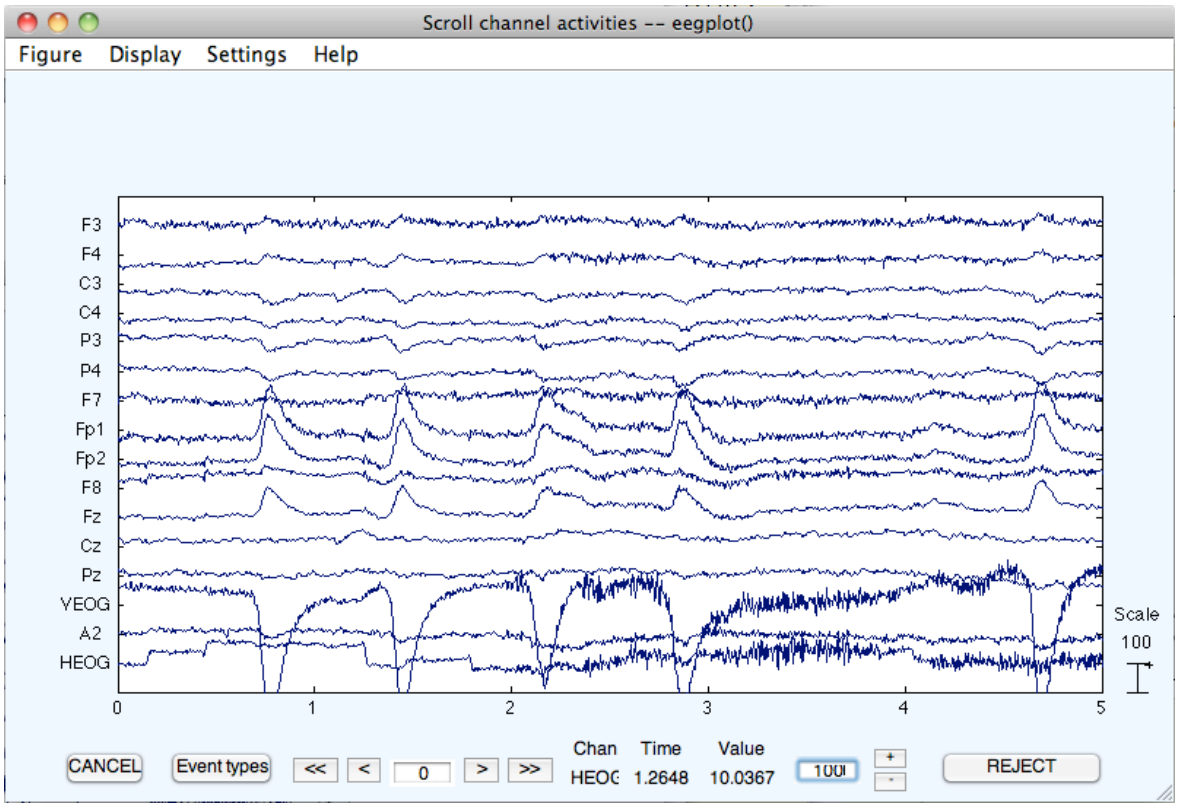
operations before or after artifact detection. Whether you do it before or after will depend on what you are trying to achieve. For example, creating a bipolar VEOG channels (as in the first example of channel operations) can make it easier to find and reject blinks, so this is usually done prior to artifact detection. In contrast, re-referencing to the average of the two earlobes won't usually change the results of artifact detection, so you can do it either before or after artifact detection (or even after averaging).

13.4. Re-referencing to the average of all scalp sites

Using the average of all electrode sites as the reference (called the *average reference* derivation) is quite popular, but it has some significant shortcomings that you should understand before using it (see Chapter 3 in *An Introduction to the Event-Related Potential Technique*). The general approach used to re-reference to the average of the two earlobes in the previous example could be used to create the average reference derivation, but it would lead to very long equations (e.g., "ch1 = ch1 – (ch1 + ch2 + ch3 +…)/16"). To make this simpler, ERPLAB provides the **avgchan** function, which computes the average of a set of channels. For example, you might want to use the average of all channels except the VEOG and HEOG channels to re-reference. To do this, you need the average of channels 1-13 and 15 (excluding channels 14 and 16, which are the VEOG and HEOG channels), which you can obtain by specifying **avgchan(1:13,15)**. The following screenshot shows how this would be implemented (you can load the equations from the file **chanops_avg_reference.txt**). But make sure you start with the original version of S1_Chan, not the version that has already been updated.



After you click **RUN**, each channel that is being recomputed is displayed in the MATLAB command window. You can then plot the data using **Plot > Channel data (scroll)**, which should look like the following screenshot.



14. Assigning Events to Bins with BINLISTER

Up to this point, we have assigned events to bins when creating the EventList structure. This is fine for very simple experiments, but it is insufficient for almost all real studies. We will now consider how event are assigned to bins for typical studies using the BINLISTER routine (see the ERPLAB User's Manual for more details).

To begin this example, select the "S1_Chan" dataset. The first step is to create an EventList. When we did this before, we used the advanced method, which allows you to provide a label for each event code and to assign each code to a bin. This time we will use the simple method, which just creates the EventList with whatever event codes or event labels are in the dataset. To do this, select **ERPLAB > EventList > Create EEG EventList – Basic**. It will put up a small window giving you a few basic options, but you should just click **Create**. You can then accept the default name for the new dataset (S1_Chan_elist). You can verify that the EventList was created by typing **EEG.EVENTLIST** in the Matlab command window.

Now we are ready to start using BINLISTER. The general approach used by BINLISTER is as follows. Using a text editor (such as the Matlab editor), you will create a text file that provides an abstract description of the kinds of event sequences that should be assigned to each bin (e.g., event code 20 preceded by event code 51 or 52 and followed 200-1500 ms later by event code 101). The abstract description for a given bin is called a *bin descriptor*, and the file containing the bin descriptors is called a *bin descriptor file*. BINLISTER reads this file, and then scans the **EVENTLIST** structure.

Each event is compared with each of the bin descriptors. If it matches one or more of the bin descriptors, the list of bins for that event is updated.

The first (and hardest) step is to create the bin descriptor file. As a reminder, here is a table showing the stimulus event codes. The event codes for responses in this experiment indicated whether the response was correct (event code = 9) or incorrect (event code = 8) rather than directly indicating whether a left-hand or right-hand response was made.

Event Code	Category	Probability	Correct Response
11	Letter	Frequent	Left Hand
21	Digit	Rare	Right Hand
112	Letter	Rare	Left Hand
122	Digit	Frequent	Right Hand
12	Letter	Rare	Right Hand
22	Digit	Frequent	Left Hand
111	Letter	Frequent	Right Hand
121	Digit	Rare	Left Hand

We will start with an example in which we collapse all the rare stimuli together into one bin (Bin 1) and all of the frequent stimuli together in a different bin (Bin 2). And we'll include only trials in which the stimulus was followed by a correct response. Here are the bin descriptors for these two bins.

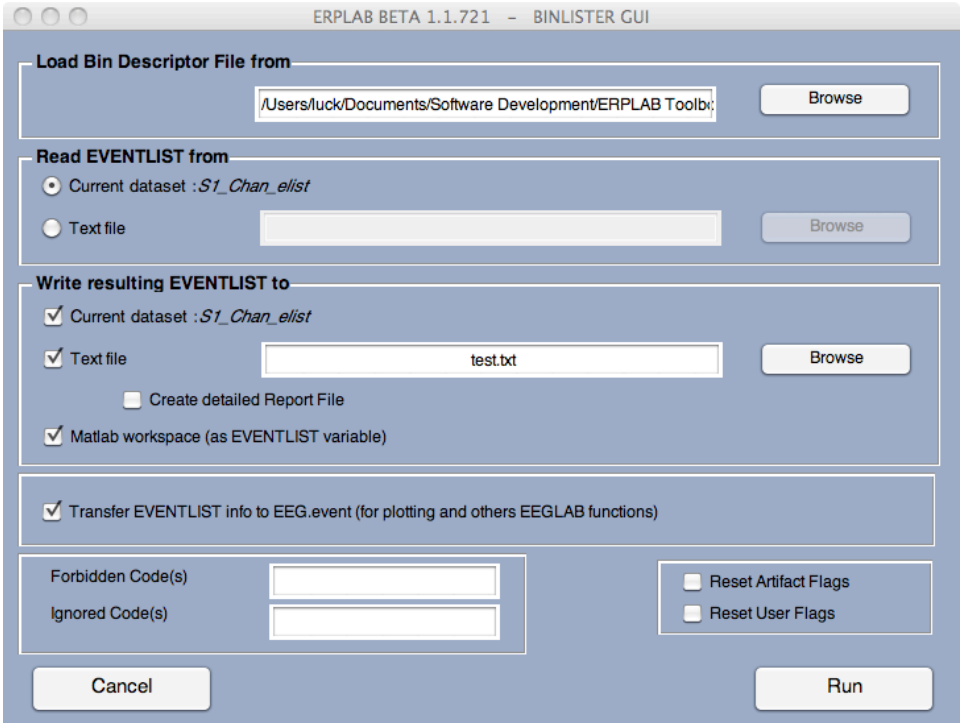
bin 1
Frequent followed by correct response
.{11;122;22;111}{9}

bin 2
Rare followed by correct response
.{21;112;12;121}{9}

Each pair of curly brackets ("{}") defines an *event list*, which specifies one or more events that must occur at a given sequential position. The event list following the dot specifies the identity of the time-locking event, and the following event lists define the events that must follow the time-locking event. In Bin 1, for example, we are looking for any frequent stimulus (event codes 11, 122, 22, or 111) followed by a correct response event code (9). Because the event list immediately after the dot contains the list of stimulus event codes, the stimulus event code will be the time-locking point. In this example, response-locked averages could be specified by place the dot after the stimulus event list and before the response event list (e.g., "{11;122;22;111}.{9}").

Bins must be specified in order, beginning with Bin 1. The line after the bin number is the *bin description*, and it is used to help you keep track of what is in each bin.

To try assigning events to bins, type the two bin descriptors shown above into a text file and save it as "binlister demo 1.txt". Then select **ERPLAB > Assign Bins (BINLISTER)** (S1_Chan_elist should be the active dataset). The window shown in the screenshot below will appear. Load the bin descriptor file that you just created, and type "test.txt" into the name of the text file for writing the results. Ordinarily you will not need to save the results to a text file, but this will help you see exactly how BINLISTER is working. Now make sure that everything else is set as shown in the screenshot, and then click **RUN**.



BINLISTER will take some time to go through all of the events in the EventList and create a new EventList with the new bin assignments. A new dataset will be created with the updated EventList. If you open test.txt with a text editor, you will see how it assigned the stimulus events to Bin 1 or Bin 2. For example, the first event code is a frequent stimulus and was therefore assigned to Bin 1. The second event code is a response, and it was not assigned to any bins. The third event code is a rare stimulus that was assigned to Bin 2. Note that event #73 was a frequent stimulus, but it was followed by an event code 8 (an incorrect response), so it was not assigned to any bins.

If you wish, you can now epoch the data, perform artifact detection, and create averaged ERPs for these two bins.

Now let's try a slightly more complicated example. This time we will eliminate trials preceded by a rare stimulus, whether the response to this stimulus was correct or incorrect (while still requiring the current stimulus to be followed by a correct response). That is, we want to see the following sequence for a frequent stimulus: frequent stimulus; correct or incorrect response; frequent stimulus; correct response (and we are time-locking to the second frequent stimulus in this sequence). Similarly, we want to see the following sequence for a rare stimulus: frequent stimulus; correct or incorrect response; rare stimulus; correct response (and this time we are time-locking to the rare stimulus).

To implement this, create a new text file called "binlister demo 2.txt" and put the following event descriptors in it:

bin 1
Frequent preceded by frequent and followed by correct response
{11;122;22;111}{8;9}.{11;122;22;111}{9}

bin 2
Rare preceded by frequent and followed by correct response
{11;122;22;111}{8;9}.{21;112;12;121}{9}

Run BINLISTER using this new bin descriptor file. If the active dataset is the one you created by running BINLISTER in the previous example, you will get a warning message asking you if you wish to overwrite the previous bin assignments (you can say yes). If you look at the EventList in test.txt, you will see that stimuli preceded by a rare stimulus (and the response to the rare stimulus) are no longer assigned to any bins.

It is often desirable to specify the time period of a response relative to a stimulus. You can do this in the bin descriptor file by using the **t<start-stop>** syntax. We could change our first example to look for a stimulus followed by a response with 200-1000 ms, as follows:

```
bin 1
Frequent followed by correct response
.{11;122;22;111}{t<200-1000>9}

bin 2
Rare followed by correct response
.{21;112;12;121}{ t<200-1000>9}
```

When you specify a time period for an event list, this is called a *time-conditioned event list*. Ordinarily, the events in the EventList structure must be in exactly the order specified by an event descriptor to be considered a match. For example, when the event descriptor is "{11;122;22;111}{8;9}.{21;112;12;121}{9}", the following sequences of event codes would be successfully assigned to this bin: 11->8->21->9, 22->9->12->9, 122->9->12->9. However, if the subject accidentally pressed the response button twice, yielding the sequence 11->8->8->21->9, this would not match the event descriptor, and this sequence would not be assigned to the bin.

However, when a time-conditioned event list, other event codes can be present without disrupting the assignment of an event to a bin. With the event descriptor ".{11;122;22;111}{t<200-1000>9}", the sequence of event codes 11->1->9 would match as long as the 9 occurred 200-1000 ms after the 11.

Imagine that subjects in an experiment occasionally made the incorrect response and then followed this with the correct response. For example, you might see the sequence 11->8->9 (frequent stimulus followed by incorrect response followed by correct response). This would still match the bin descriptor ".{11;122;22;111}{t<200-1000>9}" as long as the 9 occurred 200-1000 ms after the stimulus. To avoid this situation, you can use the following bin descriptors:

```
bin 1
Frequent followed by correct response without incorrect response
.{11;122;22;111}{~t<200-1000>8}{t<200-1000>9}

bin 2
Rare followed by correct response without incorrect response
.{21;112;12;121}{~ t<200-1000>8}{t<200-1000>9}
```

The tilde (~) character means "not". Thus, in this example, we are looking for a stimulus that is not followed by an 8 between 200 and 1000 ms and is followed by a 9 between 200 and 1000 ms.

If you would like to save the reaction times (RTs) associated with each trial of each type, you can do this by adding **:rt<"variable_name">** following the event codes associated with responses. For example, you can add this to our current example in the following manner:

```
bin 1
```

Frequent followed by correct response without incorrect response
.{11;122;22;111}{~t<200-1000>8}{t<200-1000>9:rt<"Frequent RT">}

bin 2
Rare followed by correct response without incorrect response
.{21;112;12;121}{~ t<200-1000>8}{t<200-1000>9:rt<"Rare RT">}

In bin 1, this will cause the amount of time between the time-locking event and event code 9 to be stored to a variable named "Frequent RT". In bin 2, this will cause the amount of time between the time-locking event and event code 9 to be stored to a variable named "Rare RT". These variables are stored within the bdf field of the EVENTLIST variable. Three fields are used. EVENTLIST.bdf.rtindex stores the number of variables (e.g., 2 in this example). EVENTLIST.bdf.rtname stores the names of the variables. EVENTLIST.bdf.rt stores the individual RT values for each variable.

15. Exporting and Importing EventLists to Combine Artifact Rejection and Artifact Correction

In this section, we will show how EEGLAB's artifact correction using Independent Component Analysis (ICA) can be combined with ERPLAB's artifact detection and rejection abilities. This is a somewhat complicated example, but it illustrates two important aspects of the EEGLAB/ERPLAB combination. First, it shows how (and why) to take an EventList text file that was created in one dataset, export it, and then import back into another dataset. Second, it shows how you can use artifact correction to avoid throwing out a lot of data, while still eliminating trials on which the eyes were closed when the stimulus was presented. Note that, before attempting this exercise, you should become familiar with how to perform ICA-based artifact correction in EEGLAB (see the EEGLAB documentation). Here we will only cover the aspects of artifact correction necessary for illustrating how to combine it with artifact rejection.

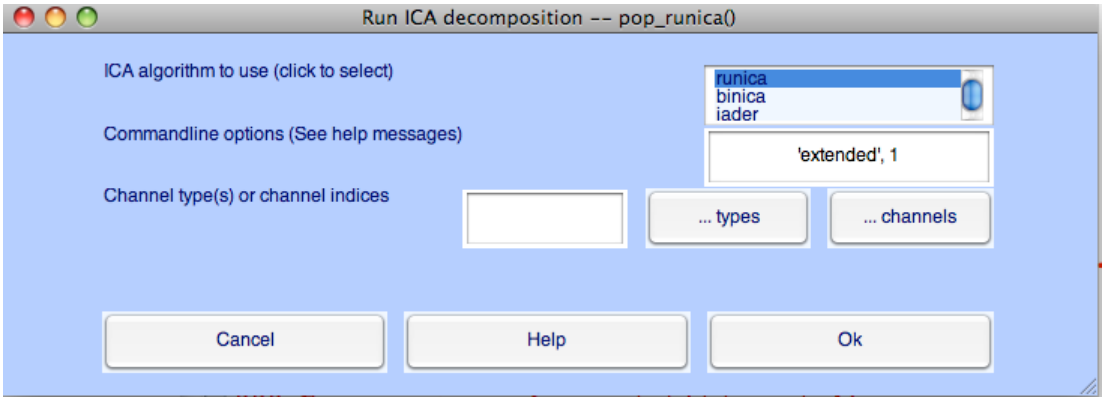
If you take a look at the first example of artifact detection in this tutorial, you will see that 29.7% of the epochs for subject S1 were detected as containing artifacts using the Moving Window Peak-to-Peak function. By scrolling through the data, it is obvious that many of these were detected due to blinks occurring during the epoch. Because the activity generated during a blink is stereotyped, a single component can usually be identified and removed to eliminate the influence of the artifact. However, the removal of a blink component does not mean that we should necessarily include all corrected trials in our analysis. For example, if the participant blinks during the presentation of a stimulus, this trial that should not be included because a participant can't see a stimulus if his or her eyes are closed due to a blink. It is important to keep in mind that artifact correction can correct the artifactual voltage associated with an ocular artifact, but it cannot correct the change in sensory input caused by an eye blink or eye movement.

The tricky part of combining artifact rejection and artifact correction is that, once the artifact correction has been performed, it is impossible to detect which trials contained a blink at the time of the stimulus. Thus, we need to perform artifact detection on uncorrected data and merge this information with the corrected EEG. This is made even more complicated by the fact that we detect artifacts after epoching the EEG, but artifact correction is usually best performed on the continuous EEG. Accomplishing this involves the following set of steps:

- Use ICA to create a continuous EEG file in which the blinks have been corrected

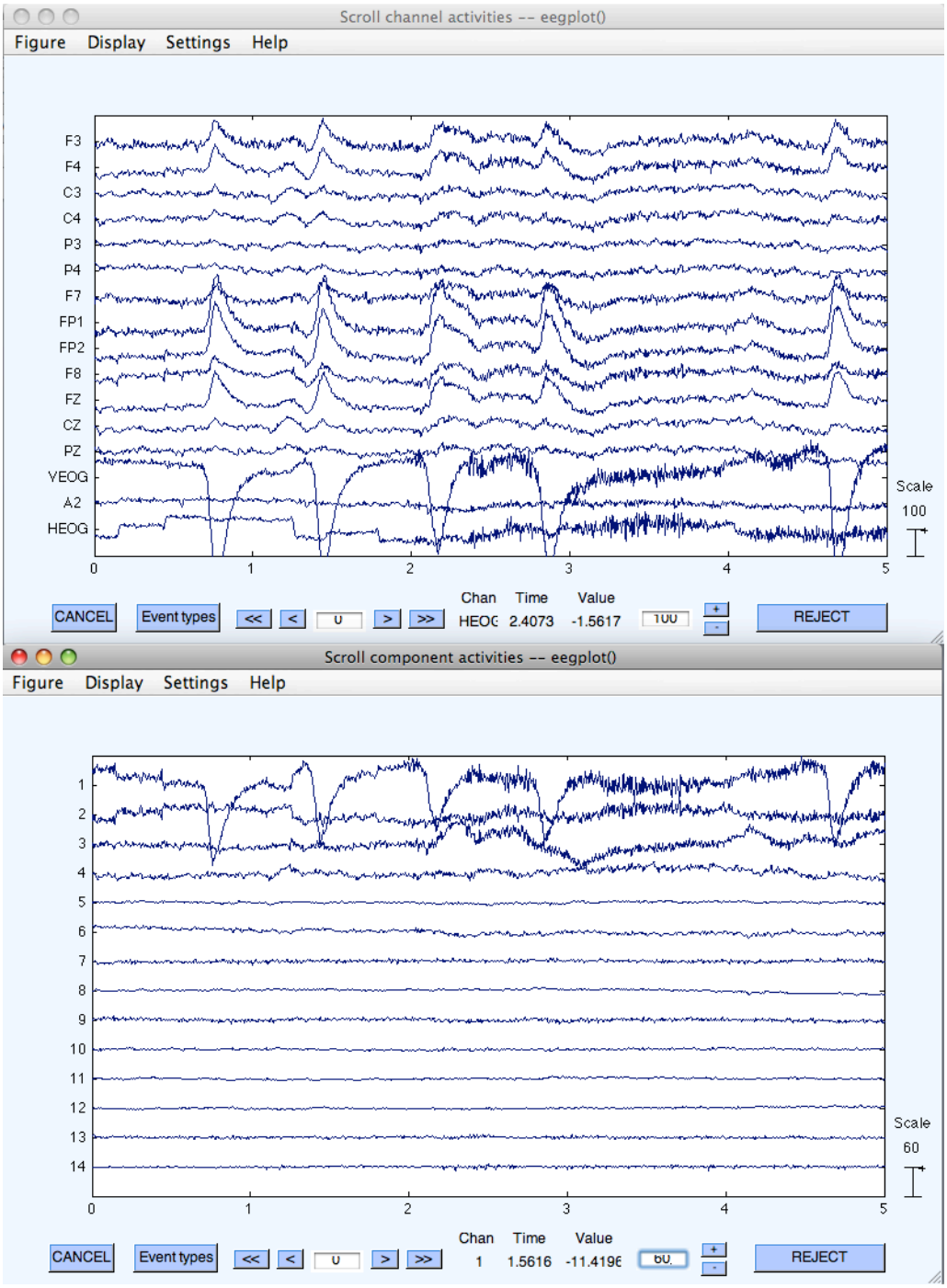
- Using the uncorrected, epoched EEG file, find trials with a blink at the time of the stimulus
- Save the EventList from this file, which marks which events contained the artifacts
- Import the EventList that marks the artifactual trials into the corrected continuous EEG file
- Epoch and average this corrected EEG file

To start, you will need to load the file **S1_Chan.set**, which was created earlier. Because running ICA will take a fair amount of time (more or less so depending on the specifics of your computer), be prepared to let it work while you go to lunch (or to sleep). To begin the procedure, select **Tools > Run ICA**. Most of the default settings can be left untouched, but it is important to only include channels that have the same reference electrode. To do this, click on **...channels**, and select all except HEOG (which is a bipolar channel created between the left and right eye channels).



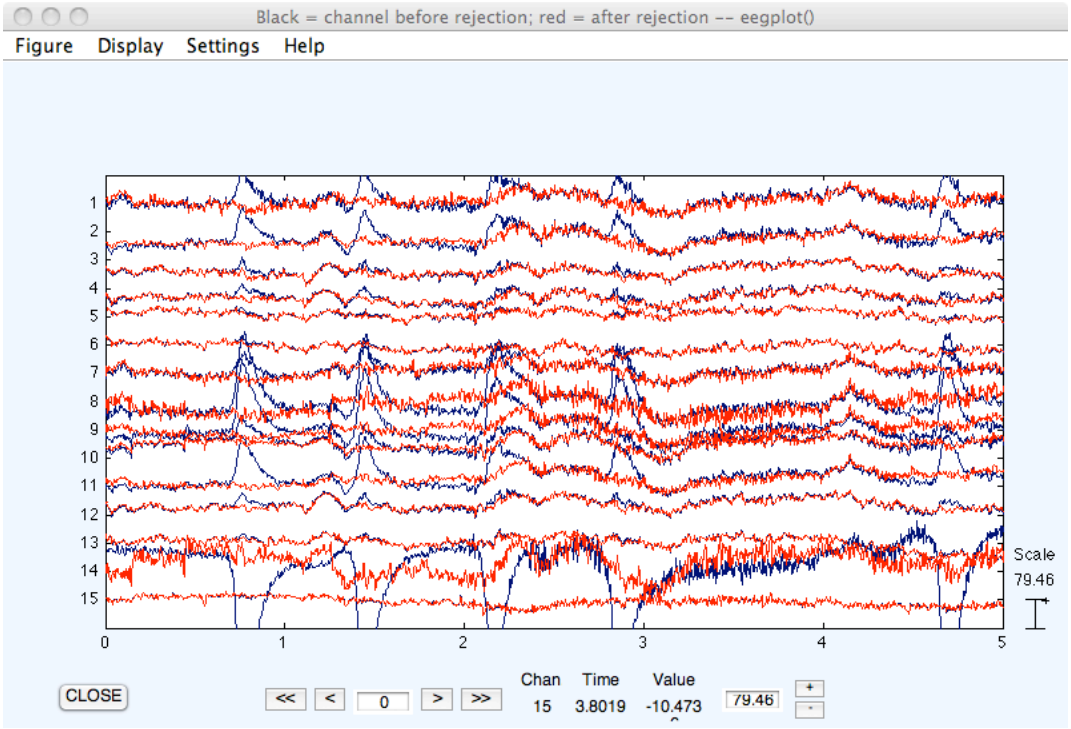
When ICA is complete, 'Done.' will be displayed in the Matlab Command Window. Make sure to save this dataset by selecting **File > Save current dataset as** and entering the filename **S1_Chan_ica.set**. You will notice in the main EEGLAB window that there is now a 'Yes' next to 'ICA weights'.

Next, you should plot the continuous EEG data and the ICA component time courses. This can be done by selecting **Plot > Channel Data (scroll)** and **Plot > Component Activations (scroll).** Align the two windows to make for easy visual comparison. You may wish to change the scaling of these plots in order to better view the data. You will see something very similar to the figure below (although likely not identical since ICA does not always return precisely the same solution). It is clear that the activations of component #1 line up in time with the occurrence of blinks in the data.



Next, we want to remove this component from the data. To do this, select **Tools > Remove Components** and enter '**1**' in the uppermost box (assuming it was component 1 that resembled blinks in your ICA results as well). After selecting **Ok**, a window will appear asking you to confirm this decision.

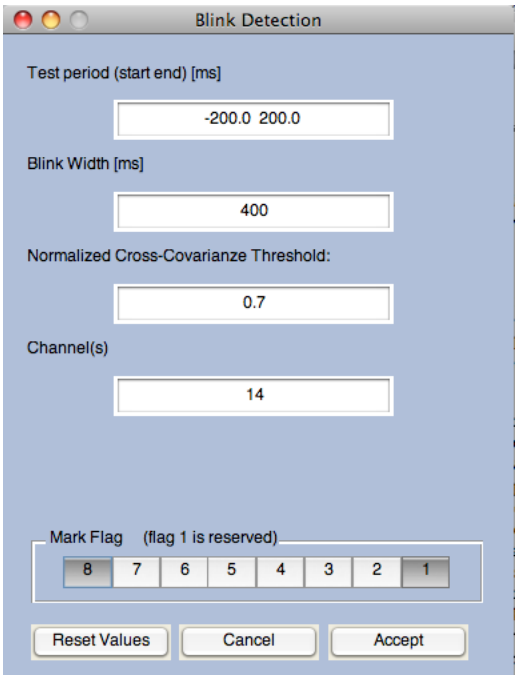
If you click on the button **Plot single trials**, you can view the original EEG superimposed with the corrected EEG that results from removing the component(s) that you have selected. It will look something like this screenshot:



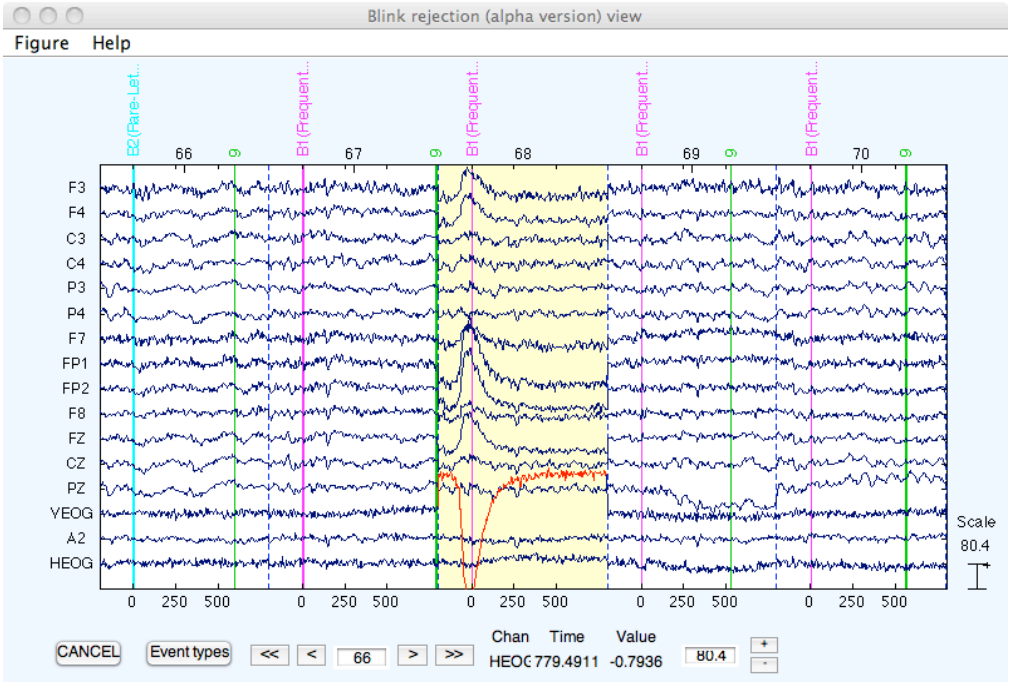
You can see that the large blinks in some channels of the original EEG (blue traces) are eliminated in the corrected EEG (red traces). If you are satisfied with the components you have selected, click on **Accept** and save this new file as **S1_EEG_Chan_icapruned.set**. We now have the corrected dataset that we will use for the final analysis.

Now that the artifactual voltages produced by the blinks have been eliminated, we need to mark the trials in which blinks occur during stimulus presentation so that these trials can be rejected. Blinks will no longer be apparent in the corrected (i.e., pruned) file, so we will use a previous version of the EEG file to determine which trials contained blinks at the time of the stimulus and export an EventList that marks those trials. To do this, select **File > Load existing dataset** and open the file **S1_EEG_elist_be** (which you created earlier in the epoching example)**.** Recall that in this version of the data, EventList information is already stored in the EEG.event structure, the bins have been defined, and the dataset has been epoched (we need the epoched version of the dataset because ERPLAB's artifact detection routines require epoched data).

We will locate epochs with blinks occurring near the stimulus by selecting **ERPLAB > Artifact Detection > Blink rejection (alpha version).** You should enter the parameters shown in the screenshot below. Because we are only concerned with marking those trials in which a blink occurred during the stimulus presentation, we will limit the test period to the time range immediately surrounding the stimulus event code, -200 to 200 ms (optimal parameters will depend on the duration of the blink; you can try varying this window to see whether it reliably detects blinks at the time of the stimulus and not blinks at other times). We will use only the VEOG (channel 14) to detect blinks. Finally, be sure to select a flag, so that we can uniquely label epochs in which blinks occurred. Here we have chosen Flag 8.



After clicking on **Accept**, you can see in the command window output that 1.6% of trials are detected as having blinks in this time period. Scroll through the EEG data viewer window that has appeared and see which epochs have been marked, as we did in the earlier examples of artifact detection. Note that most of the blinks have not been detected, because they did not occur at the time of the stimulus. However, blinks at the time of the stimulus were rejected. In particular, note epoch numbers 68, 121, 166, 221, 262. Epoch 68 is shown below, and it is clear that this blink occurred at the time of stimulus presentation, as indicated by the fact that it surrounds the stimulus event code. (Note that the bin marker has replaced the stimulus event code).



Now that we've detected the trials that we want to exclude in the uncorrected EEG data, we need to save the current EventList to a text file so that we can import it back into our ICA-corrected dataset. To do this, select **ERPLAB > EventList > Export EEG EventList to text file.** Enter **EventList_blinkdetection.txt** as the filename and click **Save**. If you open this text file in your favorite text editor, you can scroll down to the event numbered 135 (this was event #135 prior to epoching, but it was event #68 after epoching because the response events were not assigned to bins and did not generate epochs). As you can see, this was one of the events we detected. Both the 1st and 8th flags are present (Remember the 1st is always flagged when an artifact is detected for that bin. We chose also to label the artifacts detected by this step with the 8th flag).

#item	bepoch	ecode	label	onset (sec)	diff (sec)	dura (msec)	b_flags (binary)	a_flags (binary)	enable	bin
132	0	9	Corr_Resp	100.2040	594.00	0.0	00000000	00000000	1	[ ]
133	67	22	Frequent_Digit	100.9160	712.00	0.0	00000000	00000000	1	[1]
134	0	9	Corr_Resp	101.7040	788.00	0.0	00000000	00000000	1	[ ]
135	68	22	Frequent_Digit	102.3420	638.00	0.0	00000000	10000001	1	[1]
136	0	9	Corr_Resp	103.2960	953.99	0.0	00000000	00000000	1	[ ]

The final step is to import this EventList into the ICA-corrected dataset so that we can be sure to exclude the trials in which blinks occurred during the stimulus when we create ERPs from this dataset. To do this, select **File > Load existing dataset** and open the file **S1_EEG_icapruned** (or simply select It from the Datasets menu if it is still there)**.** Next, select **ERPLAB > EventList > Import EEG EventList from text file.** Choose the EventList that we just exported**, EventList_blinkdetection.txt.** A warning window will appear, asking you to confirm that you would like to replace the EEG.EVENTLIST field with this file. Select **Yes.** Finally, we will need to select **ERPLAB > Extract Bin-based Epochs** as we did previously (epoch from -200 to 800 ms, with a pre-trial baseline).

You could then average the data using **ERPLAB > Compute Averaged ERPs**. However, some other artifacts may remain in the data (e.g., movement artifacts). Thus, before averaging we will rerun the same artifact detection procedures done in the section on artifact detection. Please look back to remind your self of these parameters and run both the Moving Window Peak-to-Peak Threshold and Step-like artifact detection procedures. When the Moving Window procedure was applied to the unrejected data in the original example of artifact detection, it detected artifacts in 29.7% of epochs. Now only 10% are marked, because the blinks have been eliminated by the artifact correction process. Those epochs in which a blink occurred near to the target stimulus will also be marked as artifacts, thanks to the fact that we imported the EventList already containing these as flagged.

From here, one can proceed as before and compute average ERPs for those trials without artifacts.